NPS-CMIS-14-001



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

SEMANTIC WEB AND INFERENCING TECHNOLOGIES FOR DEPARTMENT OF DEFENSE SYSTEMS

by

Duane Davis

October 2014

Approved for public release; distribution is unlimited

Prepared for: The NPS Center for Multi-INT Studies

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From-To)*
2-10-2014	Technical Report	January 2013 – July 2013

4. TITLE AND SUBTITLE

Semantic Web and Inferencing Technologies for Department of Defense Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Duane Davis

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AND ADDRESS(ES)
Naval Postgraduate School
1411 Cunningham Road
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-CMIS-14-001

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
NPS Center for Multi-INT Studies

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The views reflected in this report are those of the author and do not reflect the official policy or position of the Department of Defense of the U.S. Government.

14. ABSTRACT

Operational commanders and intelligence professionals are provided with a continually-increasing volume of data from numerous sources. Effective utilization of this data can be hampered by difficulties in fusing different data streams for presentation, correlating related data from various sources and developing reliable summary and predictive products. An opportunity presently exists to improve this situation through the incorporation of Semantic Web technologies into Department of Defense (DOD) systems.

This report provides a didactic overview of Description Logics (DL) and their implementation in Semantic Web languages and technologies to include the mathematical properties supporting robust knowledge representation. Subsequently, the algorithms for automated reasoning and inferencing with DLs are discussed. Included in this discussion is a comparison of available Semantic Web applications for ontology development and realization or DL reasoning capabilities with real-world knowledge bases. Finally, mechanisms for applying artificial intelligence techniques to ontological DL information are presented.

15. SUBJECT TERMS
Inferencing, Description logics, Semantic Web, Ontology, OWL, RDF

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Duane Davis
Unclassified	Unclassified	Unclassified	UU	107	**19b. TELEPHONE NUMBER** ***(include area code)*** 831 656-2239

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Ronald Route President	Douglas A. Hensler Provost

The report entitled *"Semantic Web and Inferencing Technologies for Department of Defense Systems"* was prepared for the Naval Postgraduate School Center for Multi-INT Studies.

Further distribution of all or part of this report is authorized.

This report was prepared by:

Duane Davis
Research Assistant Professor
Cyber Academic Group

Reviewed by:	**Released by:**
James Scrofani, Director Center for Multi-INT Studies	Jeffrey D. Paduan Dean of Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Operational commanders and intelligence professionals are provided with a continually-increasing volume of data from numerous sources. Effective utilization of this data can be hampered by difficulties in fusing different data streams for presentation, correlating related data from various sources and developing reliable summary and predictive products. An opportunity presently exists to improve this situation through the incorporation of Semantic Web technologies into Department of Defense (DOD) systems.

This report provides a didactic overview of Description Logics (DL) and their implementation in Semantic Web languages and technologies to include the mathematical properties supporting robust knowledge representation. Subsequently, the algorithms for automated reasoning and inferencing with DLs are discussed. Included in this discussion is a comparison of available Semantic Web applications for ontology development and realization or DL reasoning capabilities with real-world knowledge bases. Finally, mechanisms for applying artificial intelligence techniques to ontological DL information are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1

A. PROBLEM STATEMENT 1
B. SEMANTIC WEB AND INFERENCING 1
C. REPORT ORGANIZATION 3
II. DESCRIPTION LOGICS AND KNOWLEDGE REPRESENTATION 5

A. DESCRIPTION LOGIC FUNDAMENTALS 5
B. DESCRIPTION LOGIC AXIOMS 10
1. Terminological and Role Axioms 10
2. Interpretations 12
3. Assertional Axioms 14
C. BASIC DESCRIPTION LOGIC REASONING 15
D. EXPRESSIVE DESCRIPTION LOGICS AND DESCRIPTION LOGIC EXTENSIONS 18
1. Expressive Description Logics 18
2. Description Logic Extensions 20
3. Trigger Rules 22
III. SEMANTIC WEB IMPLEMENTATION 25

A. METADATA 25
1. Overview 25
2. Metadata Frameworks 27
a. Extensible Markup Language 28
b. Resource Description Format 31
B. ONTOLOGIES AND THE ONTOLOGY WEB LANGUAGE 35
C. ONTOLOGY DEVELOPMENT AND MANAGEMENT 41
1. Ontology Development 41
2. Ontology Matching and Merging 45
3. Data Integration 48
D. RULE IMPLEMENTATION WITH SEMANTIC WEB ONTOLOGIES 50
IV. DESCRIPTION LOGIC INFERENCING 53

A. DESCRIPTION LOGIC REASONING FOR ONTOLOGIES 53
1. Overview 53
a. Open-World versus Closed-World Semantics 53
b. Reduction of Reasoning to Subsumption or Satisfiability 54
c. Conjunctive Query Answering 56
d. Decidability, Complexity, and Soundness, Completeness 58
2. Reasoning Algorithms 58
a. Tableau Algorithms 58
b. Automata-Based Reasoning 62
c. Resolution-Based Reasoning 64
3. Inferencing with Inductive Rules 65
B. COMPARISON OF AVAILABLE SEMANTIC WEB REASONERS 67

C. NON-STANDARD DESCRIPTION LOGIC REASONING TASKS 69
V. MACHINE LEARNING AND THE SEMANTIC WEB 71
A. OVERVIEW 71
B. INDUCTIVE LOGIC PROGRAMMING 72
C. FEATURE-BASED STATISTICAL LEARNING 76
D. RELATIONAL MATRICES AND TENSORS 78
E. ADDITIONAL SEMANTIC WEB LEARNING TECHNIQUES 79
VI. CONCLUSIONS AND RECOMMENDATIONS 83
A. CONCLUSIONS 83
B. RECOMMENDATIONS FOR FUTURE WORK 84
LIST OF REFERENCES 87
INITIAL DISTRIBUTION LIST 93

LIST OF FIGURES

Figure 1. The ontology spectrum (Daconta, et al., 03) 3
Figure 2. TBox axioms describing naval force relationships 11
Figure 3. Inductive rules for mapping an interpretation to a TBox 13
Figure 4. ABox axioms for use with the TBox of Figure 3 14
Figure 5. Required complex $\mathcal{SROIQ}$ role restrictions for reasoning decidability 20
Figure 6. Example XML document representing an air contact report 29
Figure 7. A simple XML-encoded task organization for a single operation 30
Figure 8. Graphical depiction of the XML task organization of Figure 7 31
Figure 9. A simple contact report depicted as an RDF graph 33
Figure 10. RDF triples corresponding to the contact report graph of Figure 9 33
Figure 11. An RDF(S) graph describing aircraft/ordnance relationships 34
Figure 12. Graphical depiction of an OWL ontology corresponding to the RDF(S) of Figure 11 37
Figure 13. Independent ontologies applied to overlapping domains (potential overlaps indicated with dashed lines) 46
Figure 14. Ontology-based data integration techniques 49
Figure 15. Closed-world (database) versus open-world (description logic) semantics 53
Figure 16. An example Conjunctive Query expressed with first-order logic and the associated query graph 57
Figure 17. Basic tableau algorithm for satisfiability testing of $\mathcal{ALC}$ axioms 59
Figure 18. Automata-based DL reasoning about query entailment 63
Figure 19. The Inductive Logic Programming algorithm (Muggleton and Raedt, 94) 74
Figure 20. Statistical learning algorithm for DL induction 77

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. The $\mathcal{AL}$ description logic 7
Table 2. Example $\mathcal{AL}$ axioms using “Weapon”, “Aircraft”, and “Fighter” concepts and a “payload” role 7
Table 3. Common $\mathcal{AL}$ extending operations for defining complex concepts 9
Table 4. Common $\mathcal{AL}$ extending operations for defining complex roles 9
Table 5. OWL statement exemplars in XML/RDF syntax 40
Table 6. Selected OWL statements in functional syntax and their DL equivalents (W3C, 12) 41
Table 7. Ontology-matching tool comparison (Schvaiko and Euzenat, 13) 50
Table 8. Reduction of standard TBox reasoning tasks to subsumption or satisfiability 55
Table 9. ABox transformations for the $\mathcal{ALC}$ tableau algorithm 59
Table 10. Tableau algorithm ABox transformations for $\mathcal{ALC}$ extensions 61
Table 11. First Order Logic substitutions for DL axioms 65
Table 12. Comparison of commercial and open source DL reasoners 68

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

Traditional internet technologies, including the World Wide Web (WWW, W3), facilitate the access and presentation of networked data. These technologies have obvious applications in both classified and unclassified government systems, and make volumes of potentially useful information available to operational commanders and decision-makers. This information includes raw and annotated data from tactical, strategic, and national sensors; composed and analytic products derived from various data sources; and operational information about friendly assets, just to name a few.

As the amount of available data explodes; however, it becomes more and more difficult to utilize it effectively. Access to hyperlinked documents and web-accessible data repositories does not provide the end user any contextual background or insight into how the information relates to information from other sources. Additionally, it can be difficult to separate useful information from digital clutter. Search engines can locate information based on keywords, but they have no ability to tailor results according to an "understanding" of the encountered data.

In this context, traditional distributed data storage has three significant shortcomings. First, it is difficult to efficiently separate useful data from digital clutter—imagine the difficulty in locating a specific contact of interest in hundreds of hours of unmanned air vehicle imagery over thousands of square miles. A second issue is efficient data fusion. Ideally, we would like to process data from multiple sources and locations so that redundant information is eliminated and related data is correlated. Finally, even after sorting through and fusing the data, we would like to be able to automatically draw conclusions and make predictions based on the available information.

B. SEMANTIC WEB AND INFERENCING

Semantic Web technologies provide not only access to data, but also access to contextual information that allows for its interpretation as well. Specifically, the Semantic Web uses ontologies, taxonomies, data models, and other tools to describe content characteristics and relationships. The mathematical rigor of Semantic Web

constructs provides for data discovery and utilization by networked applications and also allows for automated inferencing to derive new information and draw conclusions from distributed information.

Realization of the Semantic Web has two crucial requirements. The first is a set of standardized means of representing information. Towards this end, the World Wide Web Consortium (W3C) has approved languages such as the Resource Description Format (RDF), its extension RDF Schema (RDF(S)), and Web Ontology Language (OWL) that will be discussed in this work. The second essential element involves reasoning about represented data. Formal logic, and specifically Description Logics (DL), has received significant research attention in support of this requirement (Rudolph, 11).

Traditional distributed technologies provide access to information as opposed to knowledge. This means that data is accessible, displayable, and available for manipulation, but there is no basis for more than cursory understanding without human intervention and analysis. Semantic Web technologies, on the other hand, express meaning along with the data by adding formal semantics (Daconta, et al., 03). Formal semantics allow for the development of knowledge bases (KB) that utilize metadata to place information in context, describe relationships, make interpretations and draw conclusions in a mathematically rigorous way (Kashyap, 04). This mathematical rigor, along with recognized standards allows for the expressed knowledge to be machine read and computationally processed in ways that support automation, integration and reuse of data.

Representation of information in a form that effectively conveys knowledge requires more than simple markup of the data comprising the information. It requires a model or language that is capable of representing strong semantics about the data. (Daconta, et al., 03) uses an "Ontology Spectrum" as depicted in Figure 1 to rank various data expression mechanisms relative to one another. Traditional database techniques including Schemas, Entity-Relationship (ER) Models and Extended Entity-Relationship (EER) Models are on the lower end of this spectrum, while logical forms including Description Logic, First Order Logic, and Modal Logic are on the upper end.

Much of what we might think of as the Semantic Web technology, particularly the aspects that support automated reasoning and inferencing, are based on Description Logics (DL). DLs provide significant expressive power and have been a focus of knowledge engineering research for some time. In addition, there has been significant work in developing reasoning algorithms for working with DLs that can be proven to meet specific mathematical requirements (completeness, soundness, tractability, etc.).

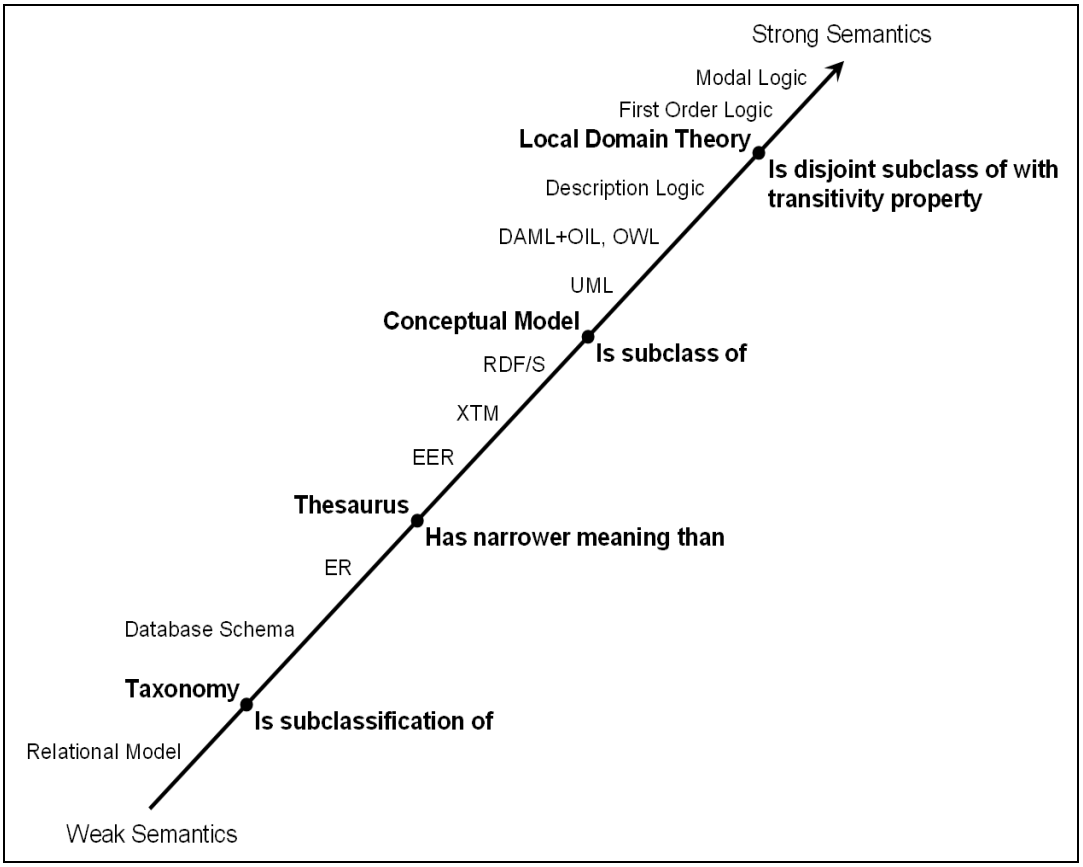


Figure 1. The ontology spectrum (Daconta, et al., 03)

C. REPORT ORGANIZATION

The remainder of this report will be organized into five sections. Chapter II contains an overview of Description Logics (DL) and their use for knowledge representation. Because DLs provide the mathematical foundation of Semantic Web technologies, understanding the Semantic Web is predicated on an understanding of DL capabilities and limitations. Chapter III provides an overview of DL implementation

through existing Semantic Web technologies and includes a discussion of the use of metadata and ontologies, metadata frameworks and standards, ontology matching and data integration, and finally, implementation of inductive rules within ontologies. In addition, a short survey of ontology development and maintenance tools is provided. Chapter IV covers standard reasoning with DLs. This chapter provides a description of standard DL reasoning tasks and the prevalent algorithms by which they are conducted. Chapter V contains a discussion of machine learning as it relates to the Semantic Web with focus on machine learning algorithms that are appropriate for inferencing tasks that cannot be accomplished through standard DL reasoning techniques. Finally, Chapter VI contains conclusions and recommendations.

II. DESCRIPTION LOGICS AND KNOWLEDGE REPRESENTATION

A. DESCRIPTION LOGIC FUNDAMENTALS

DLs are a family of logic-based knowledge representation systems that fall between Propositional Logic and First Order Logic (FOL) on the Ontology Spectrum of the (Daconta, et al., 03), meaning that they are more semantically expressive than Propositional Logic, but less so than FOL. The advantages that DLs possess over FOLs involve the decidability and tractability of associated reasoning problems. Reasoning with DLs can often be done more efficiently than with FOLs, and reasoning problems are much more likely to be computationally undecidable with FOLs than with DLs (Rudolph, 11).

DLs describe individuals within a domain of interest using concepts and roles, which describe groups of individuals and relationships between individuals, respectively. For instance, "Ship" might be a concept that includes individuals (literals) such as "Antietam", "Sullivans", and "Nimitz", and "inBattleGroup" might be a role for declaring that a ship is assigned to a specific battlegroup. DL statements, or axioms, take the form of unary and binary predicates. A unary predicate is used to apply a concept to an individual. A binary predicate, on the other hand, specifies that the individual corresponding to the first argument has the role relationship with the individual corresponding to the second argument. For example, the predicate "Ship(Antietam)" applies the "Ship" role to the individual "Antietam" and the predicate "inBattleGroup(Antietam, CSG-3)" says that the individual "Antietam" has an "inBattleGroup" relationship with the individual "CSG-3". As a rule, DL roles are not reflexive, so the declaration of the example does not imply an "inBattleGroup(CSG-3, Antietam)" relationship. Every name within a DL, then, is either a literal (individual), a unary predicate (named concept), or a binary predicate (named role) from FOL (Rudolph, 11).

DL axioms describing a particular domain are typically separated into three groups: the Terminology Box (TBox) is used to define relationships between concepts, the Relational Box (RBox) is used to define properties of roles, and the Assertional Box

(ABox) makes assertions about individuals. Together, the TBox and RBox define the intensional portion of a KB while the ABox comprises the extensional portion (Haarslev, 06). Only DLs that allow complex roles (i.e., those that include operations for combining and manipulating roles) require an RBox. Not unexpectedly, those languages that do allow complex roles provide additional expressive power at the cost of increased reasoning complexity (Krötzsch, et al., 12).

For this report, discussion will focus primarily on the TBox and ABox. Adding an RBox does not fundamentally change the reasoning algorithms beyond accounting for the role operations available in the particular DL. In practice (and in much of the literature), the TBox and RBox can be implemented and considered as a single entity (Baader, et al., 07).

DLs provide operators that can be used to build complex concepts. Specific DLs are characterized (and named) according to operations that are permitted. The most commonly utilized DLs are those that fall in the Attributive Language ($\mathcal{AL}$) family. Basic $\mathcal{AL}$ allows the definition of atomic concepts and roles and provides the operations described in Table 1 for complex concept description. The $\mathcal{AL}$ does not allow for complex roles. The basic $\mathcal{AL}$ operations can be informally described as follows:

- The universal concept ($\top$), or top concept, subsumes every concept in the domain, and the top concept describes all individuals within the domain.
- The empty concept ($\bot$), or bottom concept, excludes every concept and individual of the domain.
- Atomic negation ($\neg$A) describes all individuals for which the concept is false.
- Intersections (C $\sqcap$ D) are used to describe all individuals to which both operand concepts apply.
- Value restriction ($\forall$r.C) describes all individuals that participate as the first argument of the specified role, r, with *only* individuals to which the concept specified by the operand (C) applies.
- Limited existential quantification ($\exists$r.$\top$) describes all individuals that participate as the first argument in the specified role, r, with no restrictions on the other role participant.

Atom or Operator	Description
C, D	Named (atomic) concept
r, s	Named (atomic) role
⊤	Top (universal) concept
⊥	Bottom (empty) concept
¬A	Atomic negation
C ⊓ D	Intersection
∀r.C	Value restriction
∃r.⊤	Limited existential quantification

Table 1. The $\mathcal{AL}$ description logic

As an example, consider the following axioms of Table 2 which demonstrate simple application of $\mathcal{AL}$ operations where "Aircraft", "Fighter" and "Weapon" are atomic concepts, and "payload" is an atomic role. In these examples and their corresponding definitions, it is important to note that with $\mathcal{AL}$, negation is *only* allowed for atomic concepts, and that existential is *only* universal (i.e., the operator describes all individuals that are participating in the role relationship with no restrictions on the other role participant).

Axiom	Description
Weapon	Describes all individuals to which the "Weapon" concept applies
Aircraft ⊓ ¬Fighter	Describes all Aircraft that are not Fighters
Aircraft ⊓ ∃payload.⊤	Describes all Aircraft with a payload
Aircraft ⊓ ∀payload.Weapon	Describes all Aircraft with only a Weapon payload
Aircraft ⊓ ∀payload.⊥	Describes all Aircraft with no payload

Table 2. Example $\mathcal{AL}$ axioms using "Weapon", "Aircraft", and "Fighter" concepts and a "payload" role

While $\mathcal{AL}$ is considered a minimally robust DL, its expressive power is fairly limited. It does, however, form the basis for a family of languages that is used

extensively by Semantic Web technologies. Specific $\mathcal{AL}$ extensions are specified by letters that identify the operations that they add to $\mathcal{AL}$. The most common extensions for complex concept definition are listed in Table 3 along with their identifying letter designations. These extending operations can be informally described as follows:

- Unions ($C \sqcup D$) are used to describe all individuals to which either operand concepts apply.
- Full existential quantification ($\exists r.C$) extends limited existential qualification of $\mathcal{AL}$ to describe individuals participating in the specified role with the second participant restricted to individuals described by a specific concept.
- Unqualified cardinality restrictions ($\geq n.r$ and $\leq n.r$) describe all individuals that participate in at least (or at most) role relationships of the specific type (e.g., "≤ 5.commands" describes individuals participating in five or fewer "commands" relationships) without placing any restrictions on the role's other participating individuals.
- Qualified cardinality restrictions ($\geq n.r.C$ and $\leq n.r.C$) are similar to unqualified cardinality restrictions except they only restrict the numbers for role participants of the specified concept (e.g., "≤ 5.commands.Squadron" describes individuals participating in five or fewer "commands" relationships with individuals to which the "Squadron" concept applies).
- Negation of arbitrary concepts ($\neg C$) extends $\mathcal{AL}$'s atomic negation beyond atomic concepts by allowing the negation of arbitrary concepts.
- Nominals provide a mechanism for defining enumerated concepts in a shorthand fashion (e.g., "AirWing $\equiv$ { SH-60R } $\cup$ { MH-60F } $\cup$ { EA-6B } $\cup$ { E-2C } $\cup$ { C-2A } $\cup$ { FA-18C } $\cup$ { FA-18E } $\cup$ { FA-18F }" defines the "AirWing" concept as consisting of the eight enumerated individuals).

In addition to extensions for more robust concept description, $\mathcal{AL}$-family languages can include extensions for complex role definition as well. Many of the operations previously defined for concepts can also be applied to roles—intersection, union, and negation, for instance—and the role operators of Table 4 are also included in many $\mathcal{AL}$-family languages (the *disjoint* operator can be applied to concepts as well). The role operations of Table 4 can be informally described as follows:

- Role inversion (r^-) describes the reflection of the role to which it is applied (i.e., if r(a, b) then r^-(b, a)).
- Role composition ($r \circ s$) describes all individuals, a and c, where there exists an individual, b, that links the individuals a and c through roles, r and s (i.e., r(a, b) and s(b, c) hold for some individual, b).

- Role disjointedness (*disjoint*(r, s)) describes two roles as being mutually exclusive (i.e., if the first role holds for two individuals then the second role cannot hold for those same individuals).

Operator	Description	Designation
$C \sqcup D$	Union	$\mathcal{U}$
$\exists r.C$	Full existential quantification	$\mathcal{E}$
$\geq n.r$ and $\leq n.r$	Unqualified cardinality restrictions	$\mathcal{N}$
$\geq n.r.C$ and $\leq n.r.C$	Qualified cardinality restrictions	$\mathcal{Q}$
$\neg(C)$	Negation of arbitrary concepts (complement)	$\mathcal{C}$
Nominals	Enumerated concepts	$\mathcal{O}$

Table 3. Common $\mathcal{AL}$ extending operations for defining complex concepts

Role Operator	Description	Definition	Designation
r^-	Role inverse	{ (a, b) | r (b, a) }	$\mathcal{I}$
$r \circ s$	Role composition	{ (a, c) | $\exists$j.(r(a, b) $\wedge$ s(b, c)) }	$\mathcal{R}$
disjoint(r, s)	Disjointedness	(r(a, b) $\rightarrow \neg$s(a, b)) $\wedge$ (s(a, b) $\rightarrow \neg$r(a, b))	$\mathcal{R}$

Table 4. Common $\mathcal{AL}$ extending operations for defining complex roles

The role operations of Table 4 are particularly significant extensions because they allow the definition of a number of important complex roles. Equations 1 through 5 are templates for the respective definition of symmetric, asymmetric, transitive, reflexive, and areflexive roles.

$$r \equiv r^- \quad \text{(Eq. 1)}$$

$$\text{disjoint}(r, r^-) \quad \text{(Eq. 2)}$$

$$r \circ r \sqsubseteq r \quad \text{(Eq. 3)}$$

$$\top \sqsubseteq r.\text{Self} \quad \text{(Eq. 4)}$$

$$\top \sqsubseteq \neg r.\text{Self} \quad \text{(Eq. 5)}$$

A specific $\mathcal{AL}$-family DL is specified by the letter(s) associated with the extension(s) that it includes. For example, $\mathcal{ALIEN}$ is the $\mathcal{AL}$ extended to allow role inverses, full existential quantification, and unqualified cardinality restrictions.

By convention, the $\mathcal{ALC}$ language also includes union, full existential qualification, and a few other capabilities that will be discussed later in this report (Schmidt-Schaub and Smolka, 91). This language is among the more useful DLs and serves as the basis for what are termed expressive DLs. Because they are among the more useful (and most commonly used) DLs, the DL naming convention uses the shorthand, S, to denote $\mathcal{ALC}$ languages (Rudolph, 11).

There are a number of additional extensions that are utilized by typical Semantic Web applications that will not be specifically discussed here (Krötzsch, et al., 12). In most cases, they add expressive power to the DL but do not fundamentally change the knowledge representation or inferencing paradigms.

B. DESCRIPTION LOGIC AXIOMS

1. Terminological and Role Axioms

The TBox, denoted in equations as $\mathcal{T}$, is used to define properties and definitions for concepts that will be applied to one or more domains of interest. The TBox defines the relationships and terminology for the concepts and roles as a set of axioms. There are two primary types of TBox relationships: inclusion and equivalence. Role and concept inclusion is mathematically defined using Equations 6 and 7, respectively, while role and concept equivalence is defined by Equations 8 and 9, respectively.

$$C \sqsubseteq D \rightarrow (C(a) \rightarrow D(a)) \quad \text{(Eq. 6)}$$

$$r \sqsubseteq s \rightarrow (r(a, b) \rightarrow s(a, b)) \quad \text{(Eq. 7)}$$

$$C \equiv D \rightarrow ((C(a) \rightarrow D(a)) \wedge (C(a) \rightarrow D(a))) \quad \text{(Eq. 8)}$$

$$r \equiv s \rightarrow ((r(a, b) \rightarrow s(a, b)) \wedge (s(a, b) \rightarrow r(a, b))) \quad \text{(Eq. 9)}$$

Axioms declaring equivalences between atomic concepts or roles and other concepts or roles are called definitions. In the simple example of the Figure 2, an

equivalence relationship between the concept “Helicopter” and the complex concept of an “Aircraft” that is not “FixedWing” is explicitly defined. “FixedWing”, on the other hand, is described as being included in the concept “Aircraft”, but any equivalences must be obtained through reasoning. By these definitions, one can infer that any individual that is an “Aircraft” must also be a “Helicopter” or a “FixedWing” (but not both). “NavalUnit”, “AircraftCarrier”, “SurfaceUnit”, and “AirCapable” are similarly defined. If all definitions are acyclic, that is, it is not possible for a concept on the left hand side to use itself in its own definition, then definitions can be expanded so that only atomic concepts and roles appear on the right hand side (e.g., “AircraftCarrier ≡ (Ship ⊔ Submarine) ⊓ ∃operatedBy.Military ⊓ ∃operates.FixedWing”). An acyclic TBox is said to be definitorial, because if we know what each base symbol is (i.e., those on the right sides of the expanded definitions) then the meaning of the name symbols (i.e., those on the left sides) is completely determined (Baader, et al., 07).

$$
\begin{aligned}
&\mathsf{FixedWing} \sqsubseteq \mathsf{Aircraft}\\
&\mathsf{Helicopter} \equiv \mathsf{Aircraft} \sqcap \neg\mathsf{FixedWing}\\
&\mathsf{NavalUnit} \equiv (\mathsf{Ship} \sqcup \mathsf{Submarine}) \sqcap \exists\mathsf{operatedBy}.\mathsf{Military}\\
&\mathsf{AircraftCarrier} \equiv \mathsf{NavalUnit} \sqcap \exists\mathsf{operates}.\mathsf{FixedWing}\\
&\mathsf{SurfaceUnit} \equiv \mathsf{NavalUnit} \sqcap \neg\mathsf{AircraftCarrier}\\
&\mathsf{AirCapable} \equiv \mathsf{SurfaceUnit} \sqcap \exists\mathsf{operates}.\mathsf{Aircraft}\\
&\mathsf{operatedBy} \equiv \mathsf{operates}^{-}\\
&\mathsf{disjoint}(\mathsf{operates}, \mathsf{operatedBy})\\
&\top \sqsubseteq \mathsf{operates}.\mathsf{Self}\\
&\top \sqsubseteq \mathsf{commands}.\mathsf{Self}\\
&\mathsf{commands} \circ \mathsf{commands} \sqsubseteq \mathsf{commands}\\
&\mathsf{disjoint}(\mathsf{commands}, \mathsf{commands}^{-})\\
&\mathsf{commandedBy} \equiv \mathsf{commands}^{-}\\
&\mathsf{supports} \circ \mathsf{supports} \sqsubseteq \mathsf{supports}\\
&\mathsf{supports} \circ \mathsf{commandedBy} \sqsubseteq \mathsf{supports}\\
&\mathsf{disjoint}(\exists\mathsf{operatedBy}.\mathsf{Military}, \exists\mathsf{operatedBy}.\mathsf{Civilian})
\end{aligned}
$$

Figure 2. TBox axioms describing naval force relationships

The axioms of Figure 2 also include a number of complex role definitions describing some simple command relationship semantics. The roles "operates" and "commands" are defined to be reflexive (all individuals command and operate themselves). The "commands" role is also defined to be transitive (if an individual, a, commands an individual, b, then individual a also commands any individuals that individual b commands), and asymmetric (two individuals cannot command one another); and the roles "operatedBy" and "commandedBy" are defined as the inverses of the "operates" and "commands" roles, respectively, which implicitly confers the transitive and disjointedness properties associated with the "commands" role onto the "commandedBy" role. The role "supports" is defined to be transitive as well, and the composition of the roles "supports" and "commandedBy" is included in the role "supports" (i.e., if an individual, a, supports an individual, b, and individual b is commanded by an individual, c, then individual a also supports individual c). Finally, a "disjoint" axiom is included stating that an individual cannot be operated by both "Military" and "Civilian".

2. Interpretations

Interpretations are the mathematical mechanism through which DLs are utilized, so it is important to understand the concept of interpretations in order to understand what can be inferred from a set of ABox axioms. An interpretation is a mapping between a DL description and a specific domain of interest. The domain of interest is simply the set of all individual entities with which we are concerned.

An interpretation, $\mathcal{I}$, formally consists of a domain of interpretation, Δ^I, and an interpretation function. The domain of interpretation is the set of individuals to which the DL description is being applied, and the interpretation function assigns a set $C^I \subseteq \Delta^I$ to every atomic concept C, and a set $r^I \subseteq \Delta^I \times \Delta^I$ to every atomic role r (Baader, et al., 07). Within any particular interpretation, then, each atomic concept will consist of a subset of members of the interpretation's domain, while each atomic role will consist of a subset of the cross product of the domain of interest with itself. The inductive definitions of Figure 3 are used to intuitively map individuals to the complex concepts and roles defined in the TBox.

Based on this definition, an interpretation represents a full understanding of a domain of interest in the context of a set of TBox rules. This is the case because the domain of interest is fully defined by Δ^I, and the interpretation function represents a complete mapping of TBox concepts to the domain of interest (i.e., every individual in the domain of interest to which an atomic TBox concept or role applies is accounted for in the function). This does not mean that an interpretation represents ground truth, as a potentially infinite number of interpretations can be applied to a single TBox. An interpretation does not even have to be plausible (i.e., it can contain contradictions). In fact, assessing interpretation plausibility is a foundational DL inferencing problem that has broad applicability.

$$
\begin{aligned}
&\top^I = \Delta^I \\
&\bot^I = \emptyset \\
&(\neg A)^I = \Delta^I \setminus A^I \\
&(\neg r)^I = \Delta^I \times \Delta^I \setminus r^I \\
&(C \sqcap D)^I = C^I \cap D^I \\
&(C \sqcup D)^I = C^I \cup D^I \\
&(\forall r.C)^I = \{\, a \in \Delta^I \mid \forall b.(a, b) \in r^I \rightarrow b \in C^I \,\} \\
&(\exists r.\top)^I = \{\, a \in \Delta^I \mid \exists b.(a, b) \in r^I \,\} \\
&(\exists r.C)^I = \{\, a \in \Delta^I \mid \exists b.(a, b) \in r^I \wedge b \in C^I \,\} \\
&(\geq n.r)^I = \{\, a \in \Delta^I \mid |\{\, b \mid (a, b) \in r^I \,\}| \geq n \,\} \\
&(\leq n.r)^I = \{\, a \in \Delta^I \mid |\{\, b \mid (a, b) \in r^I \,\}| \leq n \,\} \\
&\mathrm{disjoint}(C, D) = (a \in C^I \rightarrow a \notin D^I) \wedge (a \in D^I \rightarrow a \notin C^I) \\
&(r^-)^I = \{\, (a, b) \in \Delta^I \times \Delta^I \mid r^I(b, a) \,\} \\
&(r \circ s)^I = \{\, (a, c) \in \Delta^I \times \Delta^I \mid \exists j.(r^I(a, b) \wedge s^I(b, c)) \,\} \\
&\mathrm{disjoint}(r, s)^I = ((a, b) \in r^I \rightarrow (a, b) \notin s^I)) \wedge \\
&\qquad\qquad\qquad\qquad ((a, b) \in s^I \rightarrow (a, b) \notin r^I))
\end{aligned}
$$

Figure 3. Inductive rules for mapping an interpretation to a TBox

3. Assertional Axioms

An ABox, denoted in equations as $\mathcal{A}$, describes what is known about the state of the world by making assertions about individual named entities. Assertions can be either concept assertions or role assertions and can utilize any operators allowed by the specific DL. A concept assertion typically assigns a named entity to a concept, while a role assertion establishes a role relationship between two named entities. In the example of Figure 4, a number of assertions are made including ones applying the concept "Ship" to "Nimitz", "Princeton", and "Minnow" and the role "operatedBy" to pairs "(Nimitz, Military)" and "(Princeton, Military)".

```
Military(CSG-3)
Ship( Nimitz )
Ship( Princeton )
Ship( Minnow )
Submarine( Virginia )
Aircraft( MH-60S )
FixedWing( FA-18F )
operatedBy( Nimitz, CSG-3 )
operatedBy( Princeton, CSG-3 )
operates( Nimitz, FA-18F )
operates( Nimitz, MH-60S )
operates( Princeton, MH-60S )
commands( CSG-3, Nimitz )
commands( Nimitz, Princeton )
supports( Virginia, Nimitz )
```

Figure 4. ABox axioms for use with the TBox of Figure 3

We can use this ABox information to further develop and test interpretations where the domain of interest is made up of named entities from the ABox. For an example, an interpretation satisfies a concept assertion, C(a), if and only if $a^I \in C^I$. That is, if the interpretation applies the concept C to every element in Δ^I corresponding to an individual for which an ABox asserts or implies C(a), then the interpretation satisfies the concept (satisfaction of role concepts works similarly). An interpretation satisfies the

ABox if it satisfies all of the concepts and roles contained in the ABox. If the interpretation also satisfies the TBox then it amounts to a reasonable abstract view of the domain and is said to be a model for the ABox and TBox (Baader, et al., 07).

It is appropriate at this juncture to bring up two additional points. First, it is possible for the ABox and TBox to conflict. For instance, an ABox assertion "SurfaceUnit(Nimitz)" would conflict with the TBox definition of the "SurfaceUnit" concept. Second, multiple interpretations might qualify as models for the same ABox/TBox pair, and these interpretations may conflict with one another. DLs describe only what is known, so anything missing is simply unknown and can be interpreted multiple ways. In the example, an interpretation that includes the axiom "Minnow ⊑ ∃operatedBy.Civilian" satisfies the ABox, but one that includes a "Minnow ⊑ ∃operatedBy.Military" axiom also satisfies the ABox. An interpretation including both axioms, however, does not satisfy the TBox because the "∃operatedBy.Civilian" and "∃operatedBy.Military" concepts are disjoint. This is an example of open-world semantics (Baader, et al., 07). Traditional databases, on the other hand, typically use closed-world semantics, meaning that any missing information is assumed to be false. The use of open-world semantics is an important aspect of DL reasoning and its relevance to Semantic Web technologies will be made apparent later in Chapter IV.

C. BASIC DESCRIPTION LOGIC REASONING

The fundamental TBox reasoning tasks are satisfiability, subsumption, equivalence, disjointness, and classification. Relying on the TBox mechanics described in the previous section, the notions of satisfiability, subsumption, equivalence, and disjointness can be described intuitively. A concept is satisfiable if there exists at least one model interpretation with at least one entity to which the concept applies. By extension, the entire TBox is satisfiable if a model interpretation exists in which every concept applies to at least one individual. One concept subsumes another concept if the set of individuals to which subsumed concept applies is a subset of the set of individuals to which the subsuming concept applies for every model interpretation. Two concepts are equivalent if the sets of individuals to which they apply are the same for every model

interpretation. Finally, two concepts are disjoint if for every model interpretation, the intersection of the sets to which both concepts apply is empty. Notice that the requirements for satisfiability are met by the existence of a single model interpretation, while subsumption, equivalence, and disjointness require that the requirements be met by every model interpretation. Satisfiability, subsumption, equivalence, and disjointness can be more formally defined as follows:

- Concept C is satisfiable if and only if there exists a model, $\mathcal{I}$, for $\mathcal{T}$ for which $C^{\mathcal{I}}$ is non-empty.
- Concept C is subsumed by concept D (written as $C \sqsubseteq_{\mathcal{T}} D$ or $\mathcal{T} \models C \subseteq D$) if and only if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model interpretation, $\mathcal{I}$, of $\mathcal{T}$.
- Concept C is equivalent to concept D (written as $C \equiv_{\mathcal{T}} D$ or $\mathcal{T} \models C \equiv D$) if and only if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model interpretation, $\mathcal{I}$, of $\mathcal{T}$.
- Concepts C and D are disjoint if and only if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model interpretation, $\mathcal{I}$, of $\mathcal{T}$.

The final standard TBox reasoning task is classification, which determines the subsumption hierarchy of all contained concepts. This can be a computationally complex operation (consisting of n^2 subsumption checks for a TBox containing n defined concepts); however, it can be computed off-line with the results stored for later use. TBox classification is particularly useful for ontology design and visualization and is also the basis for many optimizations for other types of reasoning (Rudolph, 11).

Any of the standard TBox reasoning tasks can be accomplished through either subsumption or satisfiability (Horrocks and Patel-Schneider, 08):

- Concept C is satisfiable if and only if $C \not\equiv_{\mathcal{T}} \bot$.
- $C \equiv_{\mathcal{T}} D$ if and only if $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$.
- C and D are disjoint if and only if $(C \cap D) \sqsubseteq_{\mathcal{T}} \bot$.
- $C \sqsubseteq_{\mathcal{T}} D$ if and only if $(C \sqcap \neg D)$ is unsatisfiable.
- $C \equiv_{\mathcal{T}} D$ if and only if $(C \sqcap \neg D)$ and $(D \sqcap \neg C)$ are both unsatisfiable.
- C and D are disjoint if and only if $(C \sqcap D)$ is unsatisfiable.

This observation implies that if either satisfiability or subsumption is a decidable computational problem, then the entire set of TBox reasoning tasks are decidable. The ability to use satisfiability as the basis for reasoning about the TBox is particularly noteworthy, because it is the basis for most implemented reasoning systems.

The most fundamental form of ABox reasoning is consistency checking. In lay terms, consistency simply means that the ABox is reasonable and does not contain any contradictions either inherently (e.g., asserting both C(a) and ¬C(a)) or with the definitions of the TBox. ABox consistency is proven by the existence of an interpretation that is a model for both the ABox and TBox. As discussed previously, an acyclic TBox can be expanded so that all definition right hand sides contain only primitives. We can use the definitions from this expanded TBox to generate an expanded ABox that contains only atomic concepts. Taking this approach, consistency checking of the ABox is reduced to checking for inconsistencies in the expanded ABox (Baader, et al., 07). TBoxes with cycles cannot be expanded this way, so this method cannot be used with cyclic TBoxes. Additionally, generating the expanded TBox can be computationally expensive making other approaches attractive even for some acyclic TBoxes.

Possibly the most common ABox reasoning task is instance checking (or entailment), which is used to determine whether or not a specific assertion (concept or role) is satisfied by every model interpretation. Additional reasoning tasks include retrieval, which identifies all individuals to which a concept applies; and realization, which is used to find the most specific concept of a set of concepts (i.e., most subsumed) that applies to an individual. Consistency, entailment, retrieval, and realization can be more formally defined as follows:

- $\mathcal{A}$ is consistent with $\mathcal{T}$ if and only if there exists a model interpretation, $\mathcal{I}$, that satisfies both $\mathcal{A}$ and $\mathcal{T}$.
- $\mathcal{A}$ entails an assertion α (written as $\mathcal{A} \vDash \alpha$) if and only if every model interpretation, $\mathcal{I}$, satisfies assertion α.
- Retrieval of concept C individuals returns the set of all individuals, a, where $\mathcal{A} \vDash C(a)$ (i.e., all individuals, a, entailed by the concept C) .

- Given an individual, a, in $\mathcal{A}$ and a set of concepts, S, what is the most specific concept, C ∈ S, such that $\mathcal{A} \vDash C(a)$ (i.e., the concept in S that most narrowly describes the individual, a, in all modeling interpretations).

Just as all standard TBox reasoning discussed thusfar can be accomplished solely through reasoning about subsumption or satisfiability, all of the standard ABox reasoning tasks described above can be reduced to consistency checking as follows:

- $\mathcal{A} \vDash \alpha$ if and only if $\mathcal{A} \cup \{\neg C(a)\}$ is inconsistent.
- A naïve approach to retrieval (which can be optimized) tests every individual in $\mathcal{A}$ for entailment.
- Realization is essentially a subsumption problem. If a subsumption hierarchy has been obtained through TBox classification, realization amounts to finding the most specific concept that entails the individual.

The fact that ABox reasoning can so frequently be reduced to consistency checking is important, and it facilitates the development of versatile algorithms that can be applied to numerous problems. Algorithms often fall into one of three categories: structural subsumption, tableau algorithms, or reduction to known First-Order logic problems. Simpler DLs can often use structural subsumption algorithms (Küsters and Molitor, 05). Tableau Algorithms are more broadly applicable to more expressive DLs and are used in a number of Semantic Web reasoners (Baader and Sattler, 01). Additional approaches that have proven applicable to working with propositional and first-order logics have proven useful, as well, when DL inferencing can be reduced to known problems from other areas of artificial intelligence and knowledge management (Rudolph, 11).

D. EXPRESSIVE DESCRIPTION LOGICS AND DESCRIPTION LOGIC EXTENSIONS

1. Expressive Description Logics

DLs that allow complex role definitions are considered expressive DLs. These languages can include all of the concept operations of the $\mathcal{AL}$-family languages (and must include all of those of $\mathcal{AL}$), but provide additional extensions for defining relationships and rules for roles (Baader, et al., 07). Thus, any language that utilizes an

RBox (even if it is implemented as part of the TBox) is considered an expressive DL. Expressive DLs are almost exclusively extensions of $\mathcal{ALC}$ (or $\mathcal{S}$, for short), which by convention extend $\mathcal{AL}$ with arbitrary concept negation, union, and full existential quantification (Ortiz, 10).

Many of today's commonly used expressive DLs are sublanguages of the $\mathcal{SROIQ}$ language (Rudolph, 11). Most importantly, this language is the basis for OWL, the W3C-approved standard for developing Semantic Web Ontologies, which will be discussed later in this work.

As with other DLs of the $\mathcal{AL}$ family, the available operations are conveyed by the letters in the title. In this case, '$\mathcal{S}$' means that it is an extension of ALC. '$\mathcal{R}$' provides for limited complex role inclusion, meaning that roles can be composed for inclusion in other roles. Specific restrictions to complex role definition are required to ensure decidability of reasoning problems. The nature of these restrictions will be discussed shortly. Also included in the '$\mathcal{R}$' designation is the ability to define reflexive and disjoint roles. Of the remaining letters, '$\mathcal{O}$' means that nominals are allowed for the definition of enumerated concepts, '$\mathcal{I}$' provides for role inverses, and '$\mathcal{Q}$' means that the language supports qualified (and unqualified) cardinality restrictions.

$\mathcal{SROIQ}$ is a powerful DL, but the expressive power greatly complicates reasoning. Even with a simple DL such as $\mathcal{AL}$, however, many reasoning tasks have been shown to be EXPTIME-hard (Baader, et al., 07). Unrestricted role composition leads to undecidability for many of these tasks (Rudolph, 11). Strict partial ordering of non-simple roles ensures decidability of reasoning problems with the $\mathcal{SROIQ}$ DL and ensures that the reasoning process will terminate. A non-simple role, as formally defined in Figure 5, is one that includes a rule composition, another non-simple role, or is the inverse of a non-simple role.

To determine whether or not a KB complies with strict partial ordering, all simple roles must first be ordered so that if an atomic role is ordered below another atomic role, then so is its inverse. Based on this atomic role ordering, the compliance of a KB with

strict partial ordering of complex roles requires each role inclusion axiom for non-simple roles to comply with one of the five forms described in Figure 5 (Horrocks, et al., 06).

2. Description Logic Extensions

DLs discussed thus far are highly expressive, but their ability to express certain types of knowledge is limited. Specifically, only knowledge that is time-independent, objective, and certain can be expressed with standard $\mathcal{AL}$-family DLs (Baader, et al., 07). It is possible, however, to implement extensions that can convey this sort of information without compromising decidability. Two potential categories of extensions are those providing mechanisms for the description of concrete domains and uncertainty.

Role Restrictions in $\mathcal{SROIQ}$

Non-Simple Roles

Role r is non-simple if

- $r_1 \circ \ldots \circ r_n \sqsubseteq r$ (if $n > 1$) or
- $s \sqsubseteq r$ and s is non-simple or
- r^- is non-simple

No other roles are non-simple

Strict Partial Ordering

For all named roles s and r

- $s \prec r$ if and only if $s^- \prec r$ and
- All role inclusions are of one of the following forms:
  - $r \circ r \sqsubseteq r$
  - $r^- \sqsubseteq r$
  - $s_1 \circ \ldots \circ s_n \sqsubseteq r$
  - $r \circ s_1 \circ \ldots \circ s_n \sqsubseteq r$
  - $s_1 \circ \ldots \circ s_n \circ r \sqsubseteq r$

  where r is a non-inverse role name and $s_i \prec r$ whenever is s_i non-simple

Figure 5. Required complex $\mathcal{SROIQ}$ role restrictions for reasoning decidability

Concrete domains provide a means of bounding data. The most obvious concrete domains are numerical, but they can also be used to capture any bounded data for which relationships can be defined. Allen's Temporal Calculus for expressing temporal relationships (Allen, 83) and Regional Connection Calculus for expressing spatial relationships (Randell, et al., 92 and Bennett, 97) are two notable examples. Concrete domains can be expressed extending the DL to include the following existential predicate

restriction: $\exists(r_1,\ldots, r_n).P$. In this definition, P is a range-restricting predicate and r_1 through r_n is a chain of functional roles to which the restricting predicate is applied in order. For instance, the declaration "AirContact $\sqcap \exists$(inArea.Restricted, isActive.Radar).overlapsWith" uses a predicate taken from Allen's Interval Calculus to describe "AirContacts" that were found in a restricted area while emitting radar.

Uncertainty, that is information that may or may not be true, is another concept that is not representable using typical DL operations. Extensions have been proposed for expressive DLs that provide for the expression of role and concept uncertainty using probabilistic, fuzzy, and probabilistic knowledge.

Probabilistic information is expressed through the addition of TBox rules for conditional probabilities of the form "P(C|D) = p" and ABox assertions of the form "P(C(a)) = p" and "P(r(a, b)) = p". Reasoning with probabilistic knowledge bases involves finding upper and lower probability bounds for each concept, which amounts to an optimization problem that can be solved with linear regression (Baader, et al., 07) or other techniques. Fuzzy information describes the degree to which concepts and roles hold. This requires redefining the typical Boolean operators from the set { 0, 1 } to the range [0, 1] and redefining the DL operators accordingly (e.g., conjunction is minimum, disjunction is maximum, etc.) (Straccia, 01). Possibilistic DLs can be considered as falling between probabilistic and fuzzy approaches in that they are used to model uncertainty but use fuzzy set semantics (Baader, et al., 07). A number of inductive reasoning techniques that will be discussed, in Chapter V, can also be used to effectively model uncertainty within a KB.

A few additional extensions are worth mentioning briefly as well. Modal logic provides for the specification of dynamic information such as belief, obligation, and other types of information that can change over time. Temporal extensions are a special type of modal logic for capturing timing. Finally, default values provide for the specification of non-monotonic knowledge; that is, concepts and implications that are usually true, but can be false in some cases (Baader, et al., 07). Many of these extensions can also be captured by inductive reasoning algorithms to be discussed in Chapter V.

3. Trigger Rules

A number of DL systems provide for the definition of inductive rules that can be used to extend KB. These rules are implemented as trigger rules expressed as FOL implications of the form C → D. Trigger rules are typically expressed as Horn clauses, which restrict the form of the consequent of the implication (D) to an atomic role or concept. The antecedent of a trigger rule (C), on the other hand, can be comprised of an arbitrary number of disjunctive (possibly complex) concepts and roles. A trigger rule expresses the notion that a specific conclusion can be asserted if certain facts are known. Thus, the statement C → D is saying that if the ABox contains assertions to support the application of the concept, C for an individual, a (i.e., the ABox entails C(a)), then the concept D also applies to that individual. This provides a powerful mechanism for inductively extending a knowledge base.

Rules can be used to extend a knowledge base using the following inferencing algorithm which amounts to a three-state finite automaton:

- Match rules—find all rules with left hand sides satisfied by contents of the ABox for which no assertion for the right hand side exists
- Select rules—determine which rules to fire in a particular iteration of the algorithm.
- Make assertions—for the selected rules, add an appropriate assertion matching the rule's right hand side.

This algorithm can be repeated until the first step returns no results. The algorithm is guaranteed to complete because both the ABox and rule set are finite. Further, the results are deterministic regardless of the order of rule application. Upon completion of the algorithm, the resultant knowledge base is referred to as the procedural extension of the original knowledge base. (Baader, et al., 07)

Because of the open-world semantics of DLs, there is a subtle, but important issue regarding the implementation of DL trigger rules. In FOL, an implication is equivalent to its contrapositive (i.e., $A \rightarrow B \equiv \neg B \rightarrow \neg A$). This equivalence holds with open-world DL semantics, however we cannot assume ¬B to be true based on the absence of a positive assertion of B. Therefore, we cannot conclude ¬A unless ¬B is entailed by the database. For this reason, trigger rules cannot be implemented as inclusions in the TBox (e.g.,

Submarine ⊑ Military) although this might seem an intuitive implementation. From a mathematical standpoint, trigger rules can be implemented by extending the DL with an epistemic concept operator as described in (Baader, et al., 07). In practice, rule bases are typically implemented as an adjunct to the DL, so the intensional portion of KB is not directly impacted (Sing and Karwayun, 10).

THIS PAGE INTENTIONALLY LEFT BLANK

III. SEMANTIC WEB IMPLEMENTATION

A. METADATA

1. Overview

The explosion of web-accessible data has already been noted as a primary motivator for the development of Semantic Web technologies. To paraphrase an early description of Semantic Web potential, goals of these technologies include bringing mathematically rigorous structure to previously disorganized data; providing unified access to distributed, heterogeneous data stores and services; facilitating seamless runtime interoperability between applications; and ultimately, improving the efficiency and productivity of human-computer interaction (Berners-Lee, et al., 01).

Mathematically rigorous web data organization fosters information discovery and use, effectively making it possible for web-based agents to eliminate meaningless and irrelevant data in favor of more meaningful and important data. Unified access requires that information be accessible to disparate applications that are not aware of its existence, much less its structure and format and semantics until runtime. Essentially, a *lingua franca* of sorts for web applications will be necessary for applications to process and interpret data so that services and applications can operate together effectively. In the end, Semantic Web technologies will allow applications to process more information without a human in the loop so that the human-computer interactive experience is more efficient and productive.

These overarching goals have a number of implications for Semantic Web content. First, Semantic Web content should be able to be understood by humans and automatically processed by machines. Both of these goals are directly supported by self-describing data—that is, data combined with meta-data describing what the data is, what units and formats are used, and the relationships between various data items. All of these goals rely on standards that provide a well-defined vocabulary for creating metadata descriptions. In practice, metadata frameworks for Semantic Web technologies allow for abstraction of content semantics from syntax and structure. This allows applications to meaningfully process information without regard for storage, implementation, or display details. (Kashyap, et al., 08)

Metadata is the fundamental underpinning of Semantic Web technologies—so much so that Semantic Web content can accurately be described as the data itself combined with the associated metadata. Metadata can be divided into two primary categories: content-independent metadata and content-based metadata. Content-based metadata is typically categorized as either structural metadata or domain-specific metadata.

Content-independent metadata is information about data that does not describe the data itself. Examples might include a contact report number, a data store location (URI), or an intelligence summary author identifier. This sort of metadata does not say anything about the actual data but can be useful in organizing, locating, and classifying data.

Content-based metadata, on the other hand, describes some aspect of the actual data. Structural metadata includes all content-based metadata that describes how the data is stored and organized. Metadata of this type can be as simple as the size of a data record or file, but a more useful example might be metadata that describes the sections of an operation order to which portions of a data record apply. Data of this sort is largely independent of the domain of the data itself. Rather, it describes how the data record is arranged so that the various pieces can be parsed and applied to specific domains of interest.

Domain-specific metadata, on the other hand, describes data in the context of a particular domain of interest. Terminology and vocabulary are key aspects of this type of metadata, as it is this metadata that enables applications to actually locate and interpret relevant data. Domain-specific metadata can be further subcategorized into two further sub-categories: Intra-domain-specific metadata and inter-domain-specific metadata.

Intra-domain-specific metadata captures relationships and associations between data in a single domain. As an example, consider an air contact report for a specific type of aircraft. Intra-domain-specific metadata in the threat data domain might be used to categorize the contact according to the types of ordnance that it carries, missions that it executes, and the countries from which it operates.

Inter-domain-specific metadata, on the other hand, captures relationships between data across two or more domains. Continuing with the air contact report example, inter-domain-specific metadata might be useful in correlating this particular contact with

intelligence assessments (i.e., inter-domain-specific metadata describing associations between the threat data domain and the intelligence domain to provide additional context between the contact report and the current operation).

2. Metadata Frameworks

A metadata framework is a formal mechanism for creating metadata; associating it with actual data; and manipulating, processing, and querying it (Kashyap, et al., 08). In order to be useful, a metadata framework has a number of fairly well-vetted components: data model, semantics, serialization, and query language.

The data model defines a collection of datatypes suitable for composing abstract views of web content. Available datatypes might include strings, integers, single- and double-precision floating point numbers, URLs, and hyperlinks. In addition to atomic datatypes, data models typically provide rules and mechanisms for defining complex data types or restrictions on existing data types. For instance, the atomic integer type might be restricted to non-negative values to represent a count, or multiple atomic types might be combined to represent a geographic location (this would require range restrictions on the atomic data types as well).

A metadata framework's semantics provide the mathematical foundation for interpreting metadata. Semantics for a metadata framework are typically described in terms of model-theoretic semantics (Marker, 07). Because DLs form the basis of many metadata frameworks, the semantics of the frameworks are captured by the rules of the underlying DL.

A serialization format provides a formal specification for how the metadata is encoded. The most common serialization format for metadata frameworks is the eXtensible Markup Language (XML), but this is by convention, not necessity (XML is designed to be human understandable and machine processable, so it aligns well with Semantic Web goals).

Finally, one or more query languages are usually available so that users (including applications) can process metadata. The query language is the mechanism by which specific data is located within a document or data source.

XML and the RDF have become well-established as the preeminent metadata frameworks for the Semantic Web (Berners-Lee, et al., 01).

a. Extensible Markup Language

XML (Bray, et al., 08) stores both data and content-based metadata in a tree structure. Each node in the tree is a named element that can have named attributes and child elements. In addition, namespaces are frequently used to identify the vocabulary from which element and attribute names are drawn.

The inclusion of metadata in the form of named elements, named attributes, and namespaces make XML documents self-describing to a point. The structural requirements of the document and the actual nature of the relationships implied by the tree structure are not explicitly contained in the document, however.

XML Schema (XML(S)) (Gao, et al., 12 and Peterson, et al., 12) provides a limited mechanism for conveying semantics of compliant XML documents. The schema for an XML document defines its vocabulary and structure, and to a degree the relationships between elements. The types of relationships and properties that can be implicitly conveyed by an XML schema are primarily limited to the "part of" relationship implied by the tree structure, the "refers to" relationship of the ID/IDREF construct, the "has characteristic of" property conveyed by attributes, and the semantics inherent in the vocabulary defined by a particular schema.

Query capability is provided by XQuery (Boag, et al., 10) and XPath (Clark and DeRose, 99) as described in (Deutsch, et al., 99).

The example XML document of Figure 6 provides a simplified contact report description. The tree structure of the document is implied by the nesting of the individual elements. All element and attribute names are in the "gccs" namespace, which in combination with the governing schema (not indicated in the figure) define the domain and vocabulary. This particular report includes information about the report in the "gccs:ReportInfo" element. The element's structure makes it clear that the report information includes the unit making the report, the sensor source for the report, and the date and time of the report (using the "gccs:dtg" attribute). The portion of the document relating to the contact itself is similarly encoded in the "gccs:ContactInfo" element.

```
<?xml version="1.0" encoding="UTF-8" ?>
<gccs:Contact gccs:ID="1234">
   <gccs:ReportInfo gccs:dtg="031345ZFEB13">
      <gccs:Unit>DDG-70<gccs:Unit>
      <gccs:Source>AN/SPY-1D<gccs:Source>
   </gccs:ReportInfo>
   <gccs:ContactInfo>
      <gccs:Location gccs:uncertainty="1000m">
         <gccs:Latitude>39 52 21.3N</gccs:Latitude>
         <gccs:Longitude>127 32 15.7E</gccs:Longitude>
      </gccs:Location>
      <gccs:AirContact>
         <gccs:Altitude>15000</gccs:Altitude>
         <gccs:Heading>210</gccs:Heading>
         <gccs:Speed>350</gccs:Speed>
      </gccs:AirContact>
   </gccs:ContactInfo>
</gccs:Contact>
```

Figure 6. Example XML document representing an air contact report

The tree data structure of XML documents has a significant limitation in defining non-taxonomical relationships. The example of the Figures 7 and 8 defines responsibilities for a single operation, represented by the XML-tree's "opord:Operation" root element. The name and commander of the operation are represented using the "opord:ID" and "opord:opcon" elements, respectively, while the elements of the operation are encoded within the "opord:Tasks" element as "opord:Task" children. The subordinate units that will be assigned specific tasks are depicted under the "opord:Assigned" element as "opord:TaskUnit" elements, while the units that will be supporting the operation are included in the "opord:Supporting" element.

The limitation of the tree data structure and the workaround mechanism become evident as individual units are assigned tasks. Specifically, the "opord:assignedTo" and "opord:supporting" attributes of the "opord:TaskUnit" and "opord:SupportingUnit" elements indicate the relevant task and unit in a human-understandable way, but given no other information there is no way to definitively associate the relevant element in a machine-processable way. XML provides the ID and

IDREF datatypes to effectively extend the tree structure by which the data is encoded to a logical graph. In the example, each “opord:Task”, “opord:TaskUnit”, and “opord:SupportingUnit” element is assigned a unique ID. The “opord:assignedTo” and “opord:supporting” attributes use the IDREF datatype to reference the relevant element. In this way, complex relationships between the various elements can be captured in an unambiguous way.

```
<?xml version="1.0" encoding="UTF-8" ?>
<opord:Operation opord:opcon=“CTF1”>Op1</opord:Operation>
   <opord:Tasks>
      <opord:Task>Task1<opord:Task/>
      <opord:Task>Task2<opord:Task/>
   </opord:Tasks>
   <opord:Assigned>
      <opord:TaskUnit opord:ID=”TU1”
                      opord:assignedTo=“Task1”/>
      <opord:TaskUnit opord:ID=”TU2”
                      opord:assignedTo=“Task2”>
         <opord:Assigned>
            <opord:TaskUnit opord:ID=”TU3”
                            opord:assignedTo=“Task2”/>
            <opord:TaskUnit opord:ID=”TU3”
                            opord:assignedTo=“Task2”/>
         </opord:Assigned>
      </opord:TaskUnit>
   </opord:Assigned>
   <opord:Supporting>
      <opord:SupportingUnit opord:ID=”SU1”
                            opord:supporting=“TU4”/>
   </opord:Supporting>
</opord:Operation>
```

Figure 7. A simple XML-encoded task organization for a single operation

Although XML has the capability of expressing significant semantics, particularly when the vocabulary and structure is governed by an XML schema, it is not without shortcomings. Specifically, while it is possible to overcome the limitations of the tree structure of XML documents through the use of the ID and IDREF datatypes, this

can quickly become cumbersome in practice. Also, it is difficult or impossible to enforce relationships that might be obvious to humans. For instance, in the example "TU4" is a subordinate of "TU2", so "TU4" should be assigned only to tasks that have been assigned to "TU2". Unfortunately, there is no structural constraint that would prevent TU4's assignment to a task associated with "TU1" or some other entity.

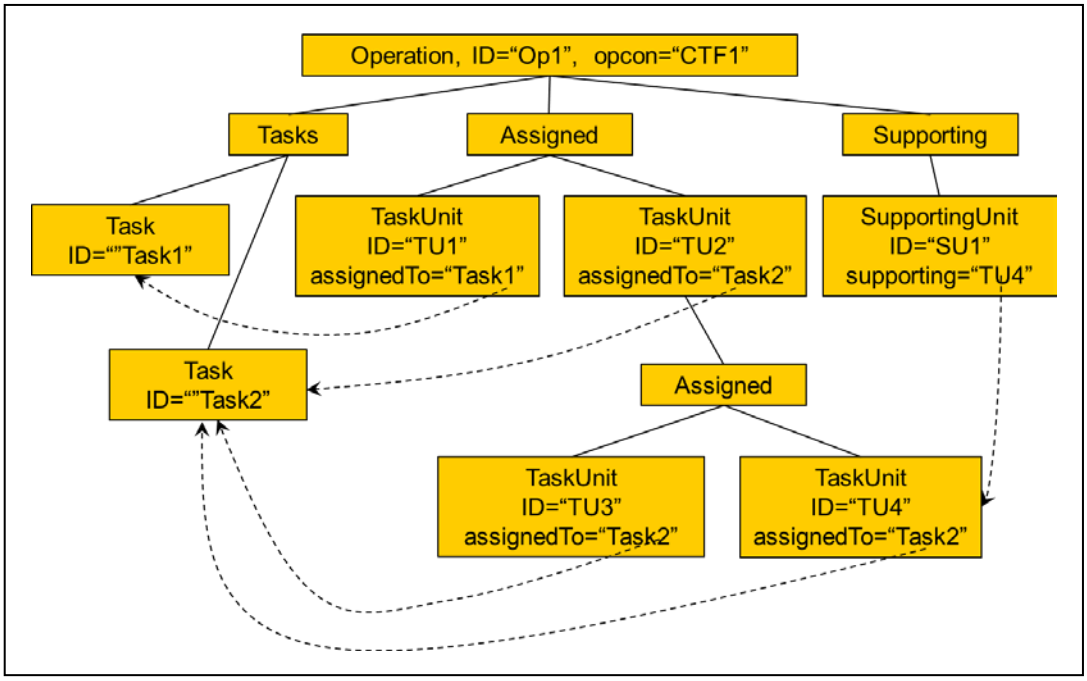


Figure 8. Graphical depiction of the XML task organization of Figure 7

Because of the issues noted (and others not discussed), XML on its own is not capable of expressing semantics with the mathematical rigor required of Semantic Web applications (Kashyap, et al., 08). It does, however, provide an underlying encoding that can be used as the basis for more expressive frameworks (Berners-Lee, et al., 01).

b. Resource Description Format

Whereas XML encodes data in a tree structure, RDF utilizes a more generic graph structure. Basic RDF components include resources, properties, and literals. Resources are the things being described and are referred to in RDF documents using a URI. A resource might be an individual data record or item, a subsection of a data record, or a collection of records (additional possibilities also exist). Properties are

specific aspects, characteristics, attributes or relationships that are used to describe resources. Literals are simply names that are used in RDF statements.

An RDF statement can be thought of as a triple of the form < subject, predicate, object >, where "subject" is a resource, "predicate" is a property that is being ascribed to that resource, and "object" is the value that is being ascribed. The object of a statement can be either a literal or another resource. Underlying encoding of RDF data can take a number of forms (the W3C recommendation calls for an XML encoding), but a triples-based approach will be used here.

As an example consider the previous contact report XML example. This example might be encoded as an RDF graph using the RDF triples of Figure 9 resulting in the RDF graph depicted in Figure 10 (note: shorthand is used for the URIs for clarity). This RDF description uses resources for the ship, the sensor, the report, the contact info, and the position; properties for associating characteristics to these objects; and literals for the concrete names and values.

The (RDF(S)) (Brickley and Guha, 04) provides facilities in the "rdf" and "rdfs" namespaces to formally define vocabularies, classes, and relationships between classes. Graphs defined in a document governed by an RDF schema must comply with the rules and structure defined in the schema in the same way that an XML document governed by an XML schema must comply with its constraints.

While both XML(S) and RDF(S) are used to constrain the content of governed documents, XML(S), is limited in its ability to convey formal semantics. RDF(S), on the other hand, provides a vocabulary that is more suited to describing relationships. In particular, RDF(S) enables the specification of subclass and subproperty relationships, property domains and ranges (subject class and object class, respectively), and other useful characteristics.

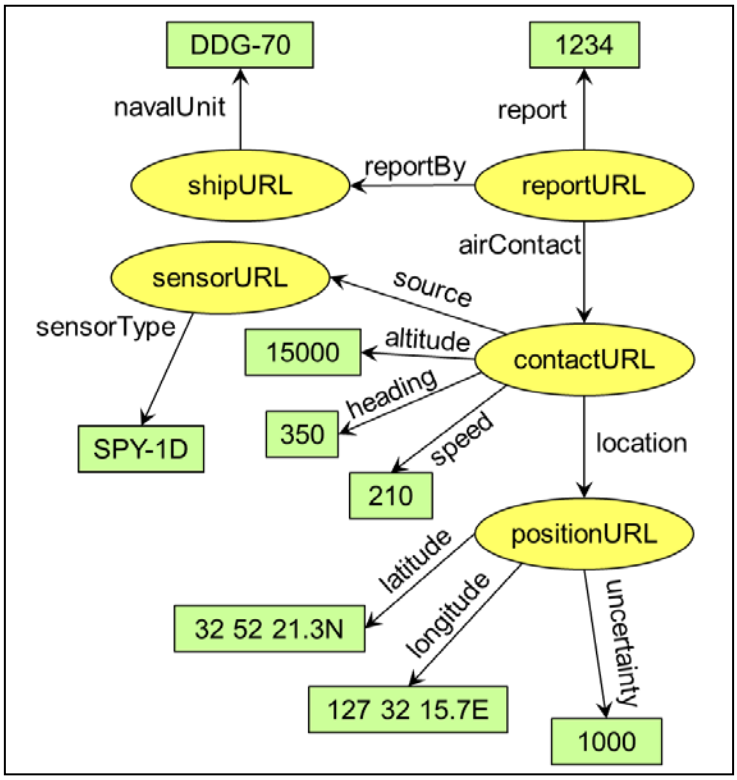


Figure 9. A simple contact report depicted as an RDF graph

```
< shipURL, navalUnit, DDG-70 >
< sensorURL, sensorType, SPY-1D >
< reportURL, report, 1234 >
< reportURL, reportBy, shipURL >
< reportURL, source, SPY-1D >
< reportURL, airContact, contactURL >
< contactURL, location, positionURL >
< positionURL, latitude, “32 52 21.3N” >
< positionURL, longitude, “127 32 15.7E” >
< positionURL, uncertainty, 1000 >
< contactURL, altitude, 15000 >
< contactURL, heading, 210 >
< contactURL, speed, 350 >
```

Figure 10. RDF triples corresponding to the contact report graph of Figure 9

The diagram of Figure 11 represents an RDF(S) description of rudimentary aircraft and ordnance relationships. In this diagram, “Aircraft” and

"Munition" are defined as subclasses of "rdfs:Class" from the RDF(S) specification. "Fighter" and "Attack" are subclasses of "Aircraft", and "FighterAttack" is a subclass of both "Fighter" and "Attack". Similar subclasses are depicted for the "Munition" class. Properties, "Airspeed", "AirOrdnance", and "GrndOrdnance", are defined as instances of the RDF(S) type "rdf:Property". The domain ("rfds:domain") classes for each property are depicted with red arrows, and range classes ("rdfs:range") are depicted with green arrows. In the example, the range for the "Airspeed" property is specified as a double precision floating point number from the XML(S) namespace ("xsd").

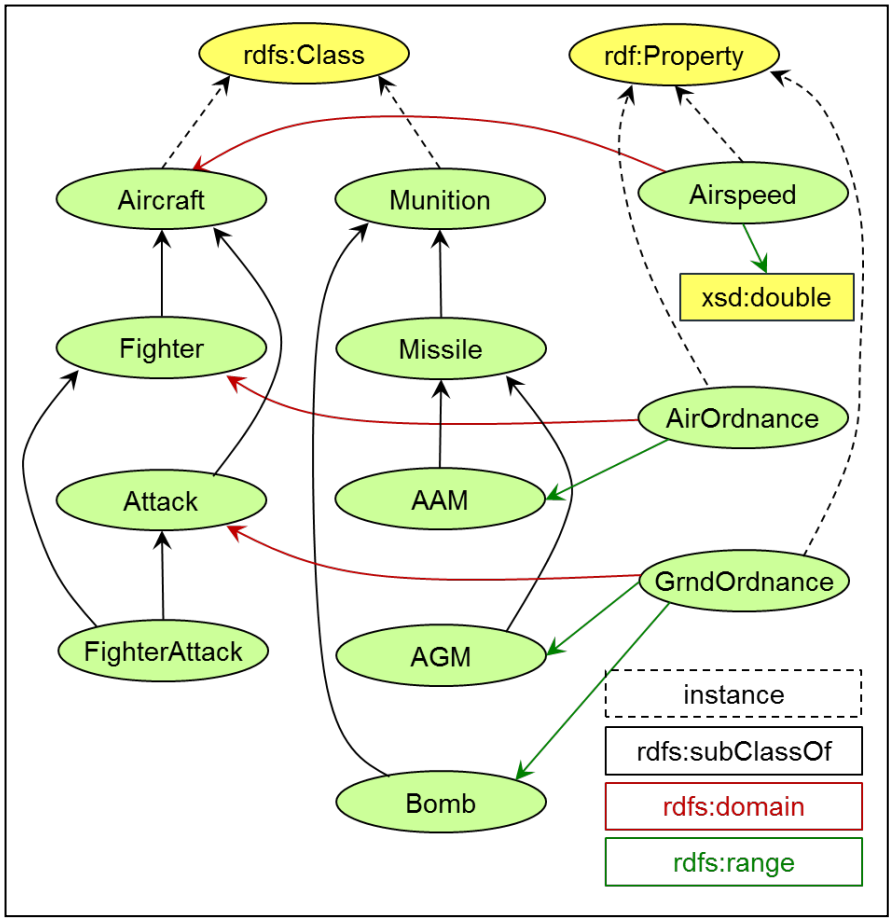


Figure 11. An RDF(S) graph describing aircraft/ordnance relationships

The SPARQL Query Language for RDF (Klyne and Carroll, 08) is the most commonly utilized mechanism for requesting information retrieval from RDF graphs. SPARQL bears a superficial resemblance to Structured Query Language (SQL); however, SPARQL is capable of significantly more robust queries than SQL. SPARQL

uses a pattern matching paradigm that takes into account the relationships defined in the RDF document and governing RDF(S) schema (Kashyap, et al., 08). For example, the SPARQL query "SELECT $ordnance WHERE { $acft Type "F-16", $acft AirOrdnance $ordnance }" will return all of the air ordnance carried by F-16 aircraft (this example assumes that a "Type" property has been defined).

Mechanisms are provided to allow for significantly more complex queries that return multiple values, place restrictions or conditions on the query, perform set operations, etc. The subclass and subproperty relationship descriptions available with RDF(S) have also facilitated the extension of SPARQL to perform many reasoning tasks appropriate for DLs (e.g., entailment) (Patel-Schneider and Simeon, 02 and Hayes, 04).

It is evident that RDF and RDF(S) provide a significantly richer mechanism for semantic expression than XML; however, they are still limited in their ability to express semantics. They are not capable, for instance, of constraining or expressing cardinality or defining conjunctive classes (Horrocks, 08). More generally, RDF, RDF(S), and SPARQL do not possess the semantic expressiveness of the basic expressive DL, $\mathcal{ALC}$, or even the "minimally interesting" DL, $\mathcal{AL}$. They do, however, provide a framework that can be used as the basis for implementing the required expressiveness.

B. ONTOLOGIES AND THE ONTOLOGY WEB LANGUAGE

Metadata descriptions with the semantic expressiveness required for the Semantic Web are frequently described as ontologies. In the context of knowledge representation, an ontology is "a specification of a conceptualization consisting of a collection of concepts, properties and interrelationships of properties" (Gruber, 93). Ontologies for the Semantic Web define the terms of interest for a particular information domain and describe the relationships among them. Semantic Web data described in terms of a specific ontology can therefore be processed alongside, compared to, and combined with other data described with the same ontology regardless of location, source, format, or composition (Horrocks, 08).

An argument can be made that many model forms might reasonably be considered ontologies. Database schemas, ER/EER Models, Unified Modeling Language (UML)

Models, XML schemas, and RDF schemas all define terminologies and describe relationships to one degree or another. As discussed previously, however, even the most semantically rich of these modeling approaches are insufficient for realizing the goals of the Semantic Web. These models can represent information and can be queried, but they do not support automated interpretation and reasoning required by Semantic Web applications.

DLs, FOLs, and modal logics provide robust description capabilities and mathematically rigorous mechanisms for reasoning. Of these, DLs have proven most applicable to Semantic Web applications. FOLs and higher logics are highly expressive semantically, but reasoning problems are often undecidable (Baader, et al., 07). Description logics, on the other hand, provide both significant semantic expressiveness and decidable reasoning (Ortiz, 10).

The Web Ontology Language (OWL) has emerged as the ontology definition mechanism of choice for Semantic Web content (Horrocks, 08). OWL is a World Wide Web Consortium (W3C) recommendation for the specification of Semantic Web ontologies. OWL allows for the definition of classes and subclasses, the association of specific properties with classes, and the definition of conjunctive classes by means of DL-based axioms. OWL is an extension of RDF meaning that OWL ontologies can be used to extend existing RDF data stores. Additionally, because OWL is an extension of RDF, SPARQL queries can be utilized to query OWL ontologies.

The OWL 1 recommendation was released in 2004 and was based on the $\mathcal{SHOIN}$ DL (Horrocks, 08) which included the operations of the basic expressive DL, $\mathcal{ALC}$, plus role hierarchy, role transitivity, role inverses, unqualified cardinality restrictions, and nominals (Baader, et al., 08). OWL 1 included three profiles, OWL Lite, OWL DL, and OWL Full, of which OWL DL provided the most broadly applicable blend of expressiveness and decidability (Kashyap, et al., 08).

The OWL 2 recommendation was released in 2008 as an extension of OWL 1 (i.e., all OWL 1 ontologies are also valid OWL 2 ontologies). OWL 2 fully implements $\mathcal{SROIQ}$ semantics described previously along with additional features for data typing (Horrocks, et al., 06 and W3C, 12).

The RDF(S) description presented earlier might be described by an OWL ontology as graphically depicted in Figure 12. OWL, however, is capable of expressing significantly more complex semantics than this simple example. Even here, one advantage of OWL might be evident. Properties in this example are associated with classes rather than associating classes (resources) with properties using "rdfs:domain" and "rdfs:range". Association of properties with classes (and instances of classes) is a more semantically accurate representation than associating classes with properties which probably do not have instances outside of the context of specific class instances.

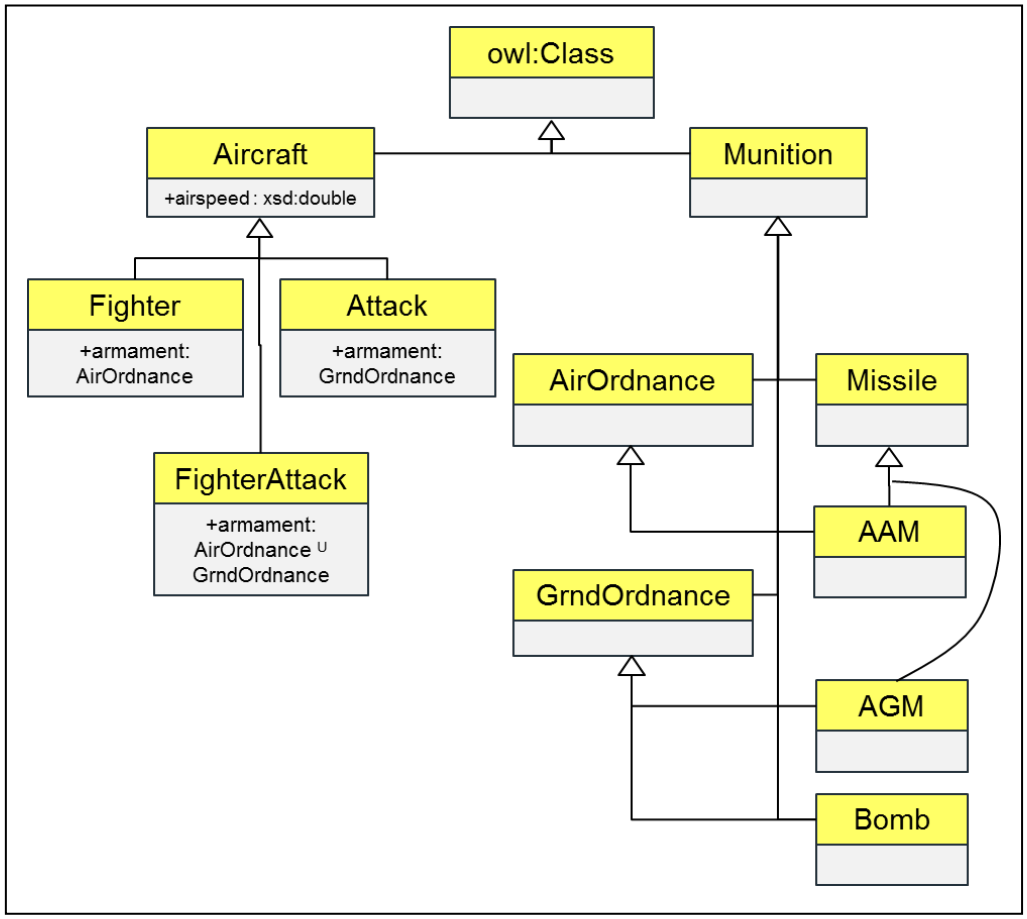


Figure 12. Graphical depiction of an OWL ontology corresponding to the RDF(S) of Figure 11

In addition to OWL 2 Full, the OWL 2 specification provides three profiles—OWL 2 EL, OWL 2 RL, and OWL 2 QL—that restrict modeling features to improve

reasoning performance. Algorithms designed specifically for reasoning with each of these profiles have been developed that execute in polynomial time (Krötszch, 12).

OWL 2 EL derives its name from the EL family of DLs which provide only existential quantification. This profile is particularly useful for large ontologies (i.e., those containing a large number of named classes and properties) (Motuk, et al., 12).

OWL 2 RL has different rules for the left and right sides of inclusion axioms. For instance, value restriction is not permitted on the right side of an axiom, while union is not allowed on the left side. This profile is the most expressive of the three. The RL acronym reflects that reasoning can be implemented using a standard rule language (Motuk, et al., 12).

OWL 2 QL is the least expressive of the three profiles, but is useful for applications that work with large sets of instance data. The OWL 2 QL profile includes most of the key features of UML and ER models that are often used with databases. The QL name reflects that queries can be implemented using a standard query language (Motuk, et al., 12).

OWL ontologies are commonly encoded with either an RDF/XML syntax or functional syntax. Table 5 provides a number of example OWL statements in the RDF/XML encoding that demonstrate some commonly utilized OWL features. These examples are not presented as complete or consistent documents (e.g., references may not refer to actual resources, examples may conflict with one another, etc.), and they should be interpreted here individually as excerpts from larger ontologies. These examples demonstrate only a small subset of the available OWL structures. A complete description of OWL components, semantics, and encodings is available in (W3C, 12).

It is worth noting that class and property definitions are not required to be contiguous, so class properties do not have to be contained within the “owl:Class” declaration and can use the “rdf:resource” attribute to extend existing classes and properties. Also, the inclusion of both “owl” and “rdf” namespace items in the examples clearly demonstrate the relationship between OWL and RDF (i.e., that OWL is an extension of RDF).

Category	Operation	Examples
Class Statements	Definition	`<owl:Class rdf:ID="Aircraft"/>`
	Subclass	`<owl:Class rdf:ID="Missile">` `  <rdfs:subClassOf rdf:resource="#Munition"/>` `</owl:Class>`
	Equivalence	`<owl:Class rdf:ID="AttackMunition">` `  <owl:equivalentClass` `  rdf:resource="#GroundMunition"/>` `</owl:Class>`
	Disjointness	`<owl:AllDisjointClasses>` `  <owl:members` `  rdf:parseType="Collection">` `  <owl:Class` `  rdf:resource="#FriendlyContact"/>` `  <owl:Class` `  rdf:resource="#HostileContact"/>` `  </owl:members/>` `</owl:AllDisjointClasses>`
	Instantiation	`<AirContact rdf:ID="air1234"/>`
Set Operations	Intersection	`<owl:Class rdf:ID="FriendlyContact">` `  <owl:intersectionOf` `  rdf:parseType="Collection">` `  <owl:Class rdf:resource="#Contact"/>` `  <owl:Class rdf:resource="#Friendly"/>` `  </owl:intersectionOf>` `</owl:Class>`
	Union & Complement	`<owl:Class rdf:ID="unknownContact">` `  <owl:complementOf` `  rdf:parseType="Collection">` `  <owl:Class>` `  <owl:unionOf` `  rdf:parseType="Collection">` `  <owl:Class rdf:resource=` `  "#FriendlyContact"/>` `  <owl:Class rdf:resource=` `  "#HostileContact"/>` `  </owl:unionOf>` `  </owl:Class>` `  </owl:complementOf>` `</owl:Class>`
	Enumeration	`<owl:Class rdf:ID="NavalUnitType">` `  <owl:oneOf rdf:parseType="Collection"/>` `  <owl:Thing rdf:about="#CVN"/>` `  <owl:Thing rdf:about="#CVN"/>` `  <owl:Thing rdf:about="#CVN"/>` `  </owl:oneOf>` `</owl:Class>`

Simple Properties	Definition	<owl:ObjectProperty rdf:ID="reportedBy"> <rdfs:domain rdf:resource="#ContactReport"/> <rdfs:range rdf:resource="#ReportingUnit"/> </owl:ObjectProperty>
	Subproperty	<owl:ObjectProperty rdf:ID="#canDeliverGround"> <rdfs:subPropertyOf rdf:resource="#canDeliver"/> <rdfs:range rdf:resource="#GroundOrdnance"/> </owl:ObjectProperty>
	Assertion	<owl:PropertyAssertion> <owl:sourceIndividual rdf:resource="#air123"/> <owl:assertionProperty rdf:resource="#airspeed"/> <owl:targetValue rdf:datatype="$xsd;unsignedInteger"> 350</owl:targetValue> </owl:PropertyAssertion>
	Negative Assertion	<owl:NegativePropertyAssertion> <owl:sourceIndividual rdf:resource="#HOPPER"/> <owl:assertionProperty rdf:resource="#reporting"/> <owl:targetValue rdf:resource="#air123"/> </owl:NegativePropertyAssertion>
Property Characteristics	Transitive & Symmetric	<owl:ObjectProperty rdf:ID="isLinkedWith"> <rdf:type rdf:resource="&owl;TransitiveProperty"/> <rdf:type rdf:resource="&owl;SymmetricProperty"/> </owl:ObjectProperty>
	Inverse	<owl:ObjectProperty rdf:ID="reporting"> <owl:inverseOf rdf:resource="#reportedBy"/> </owl:ObjectProperty>
Property Restrictions	Membership	<owl:Class rdf:resource="#AttackAcft"> <owl:Restriction> <owl:onProperty rdf:resource="#canDeliver"/> <owl:allValuesFrom rdf:resource="#GroundOrdnance"/> * </owl:Restriction> </owl:Class>
	Cardinality	<owl:cardinality rdf:datatype="$xsd;nonNegativeInteger"> * 3</owl:cardinality>
	Assignment	<owl:hasValue rdf:resource="#air123"/> *

Table 5. OWL statement exemplars in XML/RDF syntax

Table 6 provides a few examples of the relationship between statements in an OWL ontology and DLs. These examples utilize the OWL 2 functional syntax, which aligns more closely with the DL syntax of previous sections.

	Owl Syntax	**DL Syntax**
Axioms	SubClassOf(C D)	$C \sqsubseteq D$
	ClassAssertion(C a)	C(a)
	ObjectPropertyAssertion(p a b)	p(a, b)
Class Expressions	ObjectIntersectionOf(C D)	$C \sqcap D$
	ObjectUnionOf(C D)	$C \sqcup D$
	ObjectComplementOf(C)	$\neg C$
	owl:Thing	$\top$
	owl:Nothing	$\bot$
	ObjectSomeValuesFrom(p C)	$\exists p.C$
	ObjectAllValuesFrom(p C)	$\forall p.C$
	ObjectInverseOf(p)	p^-

Table 6. Selected OWL statements in functional syntax and their DL equivalents (W3C, 12)

C. ONTOLOGY DEVELOPMENT AND MANAGEMENT

1. Ontology Development

The discussion thus far might lead to one of two equally erroneous conclusions: that ontology definition is trivially easy, or that it is intractably difficult. Not surprisingly, the truth lies somewhere in the middle. Ontology development is, in fact, difficult and requires significant collaboration between domain subject matter experts and ontology implementers; however, authoring and management tools make the process manageable if used effectively.

Prior to developing an ontology, a number of considerations must be taken into account (Vidya and Punitha, 12):

- **Level of detail**: To what level of detail must the concepts and relationships be defined? What level of semantic description is required?

- **Conceptual scope**: How broad or narrow is the domain to be described (e.g., joint operational planning versus strike planning). Is the ontology a detailed component of an upper level ontology? Are the way in which the concepts and relationships are described constrained in any way?
- **Instantiation**: Will the ontology itself include instantiated individuals? Historically, all ontologies include a terminological component and applications build (or add to) the assertional knowledge base. It might be beneficial, however, to build assertional components directly into the ontology.
- **Specification language**: Numerous mechanisms for defining ontologies are available (particularly if the term is loosely applied to include taxonomies, thesauri, database schemas, etc.). This work assumes that OWL ontologies will be utilized.

Once high-level decisions requirements are determined and decisions made, ontologies are commonly developed in a stepped process along the lines of the following (Daconta, et al., 03):

1. **Acquire domain knowledge**: assemble information resources and expertise to formally describe the domain of interest.
2. **Organize the ontology**: identify the domain's principle concrete concepts and properties, identify relationships, and create abstract organizational concepts and features.
3. **Flesh out the ontology**: add concepts, relations, and individuals to obtain the required level of detail.
4. **Check the ontology**: locate and correct syntactic errors, and logical and semantic inconsistencies. This can be partially completed using reasoning techniques that check for consistency of an ontology. This is also an appropriate time for domain subject matter experts to verify the ontology.
5. **Commit the ontology**: completed ontologies must be published and made available to the applications that will rely on them.

In practice, this process will not progress as linearly as described, and might involve piecemeal development, correction, additional information gathering and consultation, etc. Further, ontology definition can still be daunting even when following a systematic development process. In fact, useful ontologies are often large enough that full human conceptualization, much less manual implementation, is impractical or impossible. The SNOMED Clinical Terms ontology contains over 400,000 named

classes (Horrocks, et al., 08). Fortunately, authoring tools are available that make ontology development and maintenance tractable.

The following provides a short comparison of a number of commonly utilized tools for ontology development and maintenance. This summary focuses on tools that support OWL ontology development and maintenance. For a more complete comparison of the most ubiquitous tools, see (Kashyap, et al., 08), (Kapoor and Sharma, 10), and (Vidya and Punitha, 12).

- **Protégé 4.1** (4.2 in beta) (CBIR, 13): Protégé is a free, open source software project implemented in Java and managed by the Stanford University Center for Biomedical Informatics Research. Protégé is a standalone system that uses a plug-in architecture to support extension. Protégé uses frames, first-order logic, and metaclasses (as opposed to a purely DL-based approach) for knowledge representation. A Protégé-OWL extension is available for support of OWL ontologies. It provides a built-in inference engine, consistency checking, and also supports selected external inference engines. It does not include support for distributed ontology development.
- **OntoStudio** (Semafora, 08): OntoStudio is a powerful ontology modeling environment commercially developed by the German Company Semafora Systems. It utilizes frames and first-order logic for knowledge representation, and plugins are available for inferencing, consistency checking, rule-based inference, and collaborative ontology development. OntoStudio does not support external inference engines but its built-in inference engine provides consistency checking and other inferencing services.
- **Ontolingua Server** (KSL, 13): Ontolingua Server is a set of tools and services developed and maintained by the Stanford University Knowledge Systems, AI Laboratory (KSL) for building of shared ontologies between distributed groups. Ontolingua utilizes the same knowledge representation mechanism as Protégé but utilizes a client-server model. Ontolingua does not include a built-in inference engine, but consistency checking and limited support for external inference engines is provided.
- **ICOM** (Franconi, 10): The Intelligent Conceptual Modeling Tool (ICOM) is an open source tool for conceptual design of information systems maintained by the Free University of Bozen-Bolzano, Italy. ICOM is based on an entity-relationship model, but utilizes DL-based knowledge representation. Consistency checking is provided, but built-in inferencing relies on connectivity with the ICOM server. ICOM does, however, support external inferencing engines.
- **IODE** (Highfleet, 13): The Integrated Ontology Development Environment (IODE) is a commercially produced modeling tool by Highfleet Semantic Technologies (previously Ontology Works) that was designed to support

ontology development for database applications. IODE is a standalone system that represents knowledge using common logic (based on first-order logic) with extensions for temporal reasoning and quantification. It includes a built-in inference engine and consistency checking, but does not support external inference engines.

- **Visual Ontology Modeler** (Sandpiper, 13): Visual Ontology Modeler™ 2.0 (VOM2) is a commercial product of Sandpiper Software for visual development of component-based ontologies. VOM2 uses DLs for knowledge representation and includes fairly robust facilities for merging, version control, and life-cycle maintenance. VOM2 does not include a built-in inference engine, but supports multiple DL reasoners and rules engines.
- **TopBraid Composer** (TopQuadrant, 11): The TopBraid™ Composer is a product of TopQuadrant, Inc. that is implemented as a standalone Eclipse plugin (Eclipse is an open-source software integrated development environment). TopBraid is designed as an enterprise-class modeling environment for developing Semantic Web ontologies and applications and represents OWL and RDF knowledge directly. It provides built-in constraint and consistency checking but does not include a built-in inference engine. It does, however, support multiple external inference engines.
- **NeOn Toolkit** (Suárez-Figueroa, 12): The NeOn Toolkit is an open source ontology engineering environment implemented as a standalone Eclipse plugin. Knowledge representation relies on Frame Logic (an alternative to DLs for ontology definition), OWL, and RDF. NeOn provides built-in constraint and consistency checking and includes a built-in inference engine. It also supports external inference engines.

Given the inherent difficulties of the ontology development process, it is not surprising that automatic ontology development is an active area of ongoing research. Three general approaches have been proposed: supervised machine learning, natural language processing, and statistical clustering.

Supervised Machine learning employs (often manually generated) positive and negative examples to "train" the tool (i.e., tune the algorithmic parameters and thresholds). The utility of machine learning approaches to automated ontology development is limited by the requirement for large numbers of training examples. Nevertheless, examples of automated and semi-automated machine-learning-based approaches to ontology development are available (Maedche, et al., 03 and Kashyap, et al., 08).

Natural language processing (NLP) has also been used as the basis for automated ontology development. This is not surprising in light of the direct application of DLs to

the field of NLP. In particular, the Suggested Upper Merged Ontology (SUMO) has produced a mapping between WordNet parts of speech and SUMO classes (Pease, 04) that can be used to extract meaning and relationships from free-form text documents (Lin and Sandkuhl, 08).

Statistical clustering and data mining have also been utilized to cull patterns and groupings from large volumes of data. Although effective at identifying related entities and visualizing data, these approaches do not lend themselves to identifying the nature of relationships or generating labels for statistically-grouped entities (Kashyap, et al., 08).

2. Ontology Matching and Merging

Experiences in the development of Semantic Web technologies have clearly demonstrated a need for ontology matching and merging. Development of ontologies in isolation has inevitably led to multiple ontologies being applied to a single domain, overlapping of domains, and definition of ontologies for different but related domains. In all three cases, differences in terminology, level of detail, definition of concepts and relationships and other factors will occur. In order to fully leverage ontologies' knowledge representation, it must be possible to align differing ontologies to enable Semantic Web applications to utilize all available data.

Figure 13 graphically depicts portions of ontologies, O_1 and O_2, which describe two overlapping domains. O_1 is an excerpt from a notional strike planning ontology and O_2 is an excerpt from a notional flight scheduling ontology. It might be obvious to a human observer that the elements "sp:FA-18E" and "sp:GBU-38" in O_1 are equivalent to "ap:FA-18E" and "ap:GBU-38," respectively, in O_2. It is also the case in this example that all individuals of the "ap:ATOEvent" class are also members of the "sp:StkElement" class (the reverse is not necessarily true). Finally, relationships might also be inferred between the "sp:assignedAC", "sp:weapon", and "sp:loadWith" properties in O_1 and the "ap:assignedUnit", "ap:loadPlan", and "ap:loadout" properties from O_2, respectively.

Taking advantage of these relationships effectively increases the size of an ontology and potentially improves the fidelity as well. In the case of multiple ontologies in a single domain, additional classes, relationships and individuals are effectively added to both ontologies. This is also the case with merging overlapping ontologies for the region of overlap, but the merge yields the additional benefit of making the relationships

between the two domains exploitable by applications as well. Similarly, formally defining the relationships between two disparate but related domains enables Semantic Web applications to exploit those relationships and use knowledge from both domains.

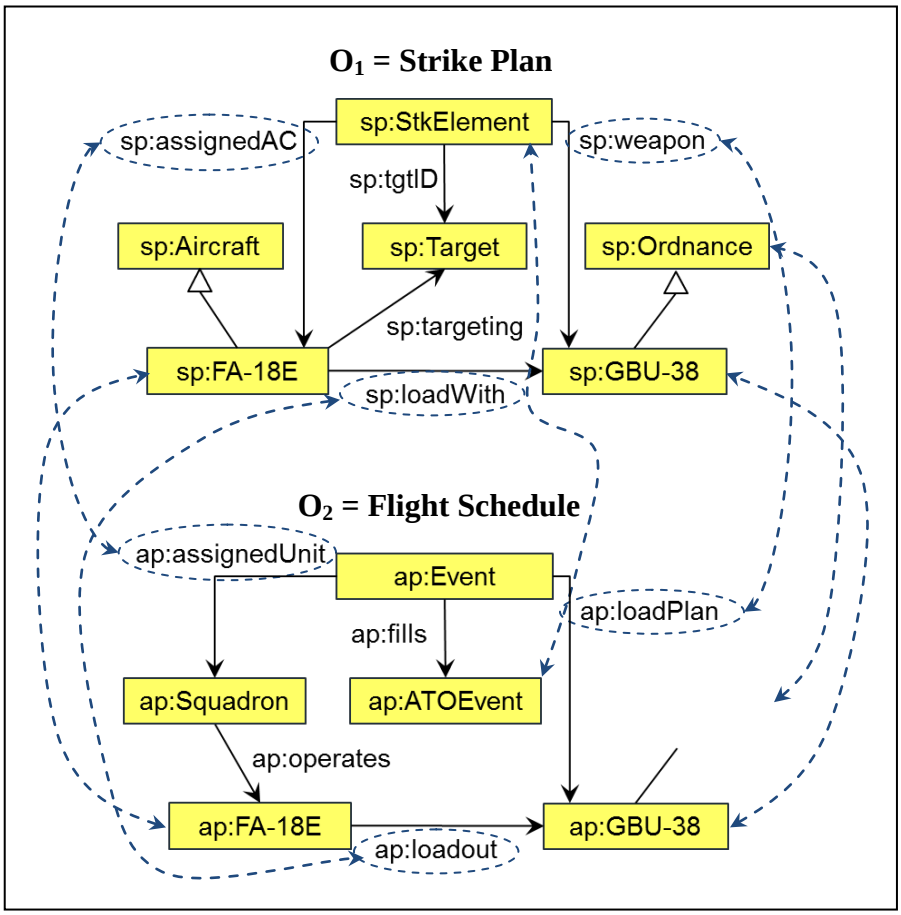


Figure 13. Independent ontologies applied to overlapping domains (potential overlaps indicated with dashed lines)

The problem of ontology matching can be formally defined as follows: given two ontologies O_1 and O_2, determine an alignment, A, defining correspondences between O_1 and O_2 where A is a set of correspondences defined as 5-tuples of the form $A_{id} = < id, e_1, e_2, n, r >$ (Shvaiko and Euzenat, 08). The first tuple element, id, is a unique identifier for the specific correspondence. Elements e_1 and e_2 are entities (classes, relationships, individuals, data values, etc.) from O_1 and O_2, respectively, to which the correspondence is being asserted. The confidence in the alignment between entities e_1 and e_2 is specified by a value, n, in the [0, 1] range. During the matching process, the confidence level is

used by the matching algorithm to determine whether or not to actually apply the presumed relation based on the application's requirements. If the relation confidence reduces to a binary value (0 or 1), then 0-confidence correspondences can be eliminated and remaining correspondences can be specified as 4-tuples of the form $A_{id} = <$ id, e_1, e_2, r > (Shvaiko and Euzenat, 13). The specific relation between e_1 and e_2 that is being asserted is specified with the variable r. Equivalence relationships are the most common assertions; however, any valid relation definable by the ontologies' DL is permissible (e.g., more general (⊒), less general (⊑), disjoint (⊥), overlapping (⊓), etc.).

Ontology matching is an area of significant ongoing research. Matching techniques are typically placed into one of four categories: terminological, structural, semantic, or extensional (Shvaiko and Euzenat, 13).

erminological matching can be either string-based or linguistic. String-based matching directly compares strings and substrings from the two ontologies. Basic string comparison techniques include prefix/suffix checking (i.e., does one string start or end with the other), edit distance (number of changes required to "transform" one string into the other), n-gram testing (common sequences of n characters in both strings), tokenization, and other manipulations that provide metrics that can be compared from string to string. Linguistic matching utilizes the linguistic characteristics of the entities being compared and leverages general or domain-specific knowledge contained in external thesauri, dictionaries or taxonomies (WordNet is a frequently utilized common knowledge source) to facilitate interpretation of individual names. Linguistic matching techniques exploit known relationships and definitions associated with ontological terms to infer relationships between the entities with which they are associated.

Structural matching analyzes the graph structures of portions of the ontologies to find similarly structured sections. Structural similarity between inner ontology nodes can be based on the similarity of their children, their leaves, their relations or some combination of the three. Graph matching is typically encoded as an optimization problem where a match minimizes some dissimilarity measurement (Kashyap, et al., 08). Alternatively, approaches that compare graphs based on similar hierarchical (is-a, has-a) characteristics or predetermined anchors (i.e., matches that are known *a priori*) have been shown to be effective (Schvaiko and Euzenat, 13).

Semantic matching utilizes model-theoretic analysis to make comparisons. These methods rely on DL- or rule-based reasoning, or other logical reasoning to deduce correspondences. Although the mathematical foundations of these approaches are well vetted, they have only recently received significant attention in research and production systems for ontology matching.

Extensional matching attempts to find relationships between instantiated instances of ontology classes. These approaches can use many of the previous techniques to identify matching or related individuals in the two ontologies.

A comparison of a few state-of-the-art ontology matching/merging systems is provided in Table 7. A reasonable conclusion to be drawn from the comparison is that no single technique is likely to prove sufficient. Rather, each of the depicted systems uses a number of techniques to achieve reasonable results. Even so, ontology matching is not an error-free process: benchmark testing documented in (Schvaiko and Euzenat, 13) yielded precision, recall, and F-measure results in the 0.8 to 0.95 range, however testing on difficult real-world problems yielded results in the 0.4 to 0.65 range. For this reason, ontology matching is not yet a fully-automated process. All of the systems summarized above generate recommended correspondences that can be either accepted or rejected by a human operator.

A more thorough comparison of each system, including benchmark metrics is provided in (Schvaiko and Euzenat, 13).

3. Data Integration

Closely related to the issue of ontology matching and merging is the problem of data integration, which deals with actual utilization of web-accessible data by Semantic Web applications. Data must frequently be drawn from databases and other types of archives that are not governed by any sort of ontology. In order for these data sources to be used by Semantic Web applications, the data must first be integrated into an ontology. The three general approaches depicted in Figure 14 have been utilized for this purpose: a single ontology approach, a multiple ontology approach, and a hybrid approach (Kashyap, et al., 08).

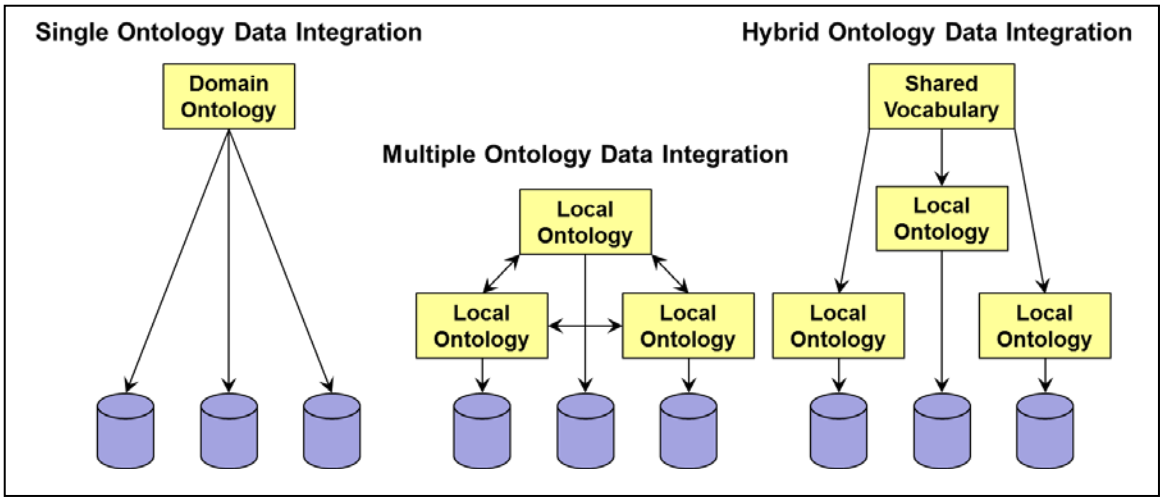


Figure 14. Ontology-based data integration techniques

With the single ontology approach, a single domain ontology is used by the application to access multiple data stores. This method has proven useful in cases where all of the data sources view the domain similarly. Even so, differences in level of granularity can make the definition of a single ontology into which all of the data can be integrated difficult. As the heterogeneity of the various data stores increases (i.e., the less they overlap) or the similarity of their views of the data decreases the definition of a single ontology for working with all of them becomes more difficult. The tendency towards a prohibitively monolithic ontology can be partially overcome through modularity, however the difficulties cannot be completely eliminated in this manner.

With a multiple ontology approach, a separate ontology is defined for each data source. The advantage of this approach is that each ontology can be designed around the specific data sources to which it will apply. In practice, however, differences in terminology and organization require significant ontology match/merge efforts to enable individual applications to utilize all of the data stores together.

The hybrid ontology approach to data integration attempts to combine the advantages of the single and multiple ontology approaches. With this approach, a single shared vocabulary (which might, itself, be defined as an ontology) is utilized for the application domain, and local ontologies based on the shared vocabulary are developed for specific data sources. Because the shared vocabulary does not have to include all of the relationships and semantics of the individual data stores, it can be more easily defined

than can a single ontology. On the other hand, availability of a common terminology eliminates the requirement for painstaking matching between the various local ontologies (the shared vocabulary makes merging implicit).

D. RULE IMPLEMENTATION WITH SEMANTIC WEB ONTOLOGIES

Recall from Chapter II on DLs that inductive trigger rules can be implemented as a DL extension. These rules take the form of implications where the truth of a consequent clause is implied by the truth of an antecedent clause: $C \rightarrow D$. In order to be useful, rules are typically constructed as Horn clauses, meaning that the consequent (D) consists of an atomic concept or role and the antecedent (C) is a disjunctive series (0 or more) of complex and/or atomic concepts and roles.

System	Terminological	Structural	Semantic	Extensional
Falcon	I-SUB, Virtual documents	Structural proximity, Clustering		Object similarity
DSSim	Tokenization, WordNet	Leaf-based graph similarity	Rule-based fuzzy inference	
RiMOM	Edit distance, Vector distance, WordNet	Similarity propagation		Vector distance
ASMOV	Tokenization, String equality, WordNet	Iterative fixed-point computation, Hierarchical	Rule-based inference	Object similarity
Anchor-Flood	Tokenization, String equality, WordNet	Internal, external similarity, Anchors		
AgreementMaker	Edit distance, Substrings, WordNet	Descendant, sibling similarity		

Table 7. Ontology-matching tool comparison (Schvaiko and Euzenat, 13)

Rules can be added to an ontology as an additional layer that effectively extends the semantic expressiveness of the ontology. For instance, the trigger rule of the following expression asserts that if an individual, a, is operated by an individual, b, and individual b is in hostilities with a third individual, c, then individual a will attack individual c:

$$\text{Country}(a) \wedge \text{Unit}(b) \wedge \text{Country}(c) \wedge \text{operates}(a, b) \wedge \text{inHostilities}(b, c) \rightarrow \text{willAttack}(a, c)$$

The relationship conveyed by this example is difficult or impossible to express with most DLs and is not possible with the $\mathcal{SHOIN}$ or $\mathcal{SROIQ}$ DLs implemented by OWL 1 and OWL 2, respectively.

A number of relevant observations can be made regarding the utilization of Horn-clause-based rules with ontologies. First, rules can be thought of as an extension of an ontology framework that provides an additional expressive operation (Baader, et al., 07). Also, rules are applied to individuals to make new assertions about those individuals: the rule itself is universally quantified (i.e., it applies to every set of applicable individuals), but it is only executed (i.e., assertions are only made) in cases where the antecedent conditions hold for specific individuals. Further, since the knowledge base is finite, there are finite possible rule applications. Thus, once every possible rule execution has been triggered, the knowledge base will have been deterministically expanded.

The Semantic Web Rule Language (SWRL) (Horrocks, et al., 04) is a W3C member submission that extends OWL by adding Horn-like rules from the Rule Markup Language (RuleML). Many of the ontology development and management tools previously discussed are capable of working with SWRL rules, as are most of the reasoning tools that will be discussed in Chapter 4.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DESCRIPTION LOGIC INFERENCING

A. DESCRIPTION LOGIC REASONING FOR ONTOLOGIES

1. Overview

a. Open-World versus Closed-World Semantics

Formal reasoning with DLs is heavily influenced by their open-world semantics, so the distinction between open-world and closed-world semantics bears further discussion. With the closed-world semantics of traditional databases (typically characterized by ER, EER, or UML models), missing information is treated as false. The database schema, therefore, can be viewed as a set of constraints on the contained data. With the open-world semantics of ontologies, missing information is considered unknown. The axioms of the ontology, therefore, can be viewed as inference rules for expanding the KB (Horrocks, 08).

Consider the examples of Figure 15. The simplified EER diagram on the left specifies that "MilitaryAcft" and "CivilianAcft" are both subtypes of the "Aircraft" supertype and that "Missile" and "Bomb" are subtypes of "Weapon". In order to add a record specifying that an individual aircraft, "FA-18_172396", is armed with an "AIM-120X", it must already be known (i.e., present in the appropriate database tables) that "FA-18_172396" is an "Aircraft", that it is a "MilitaryAcft", and that "AIM-120X" is a "Weapon".

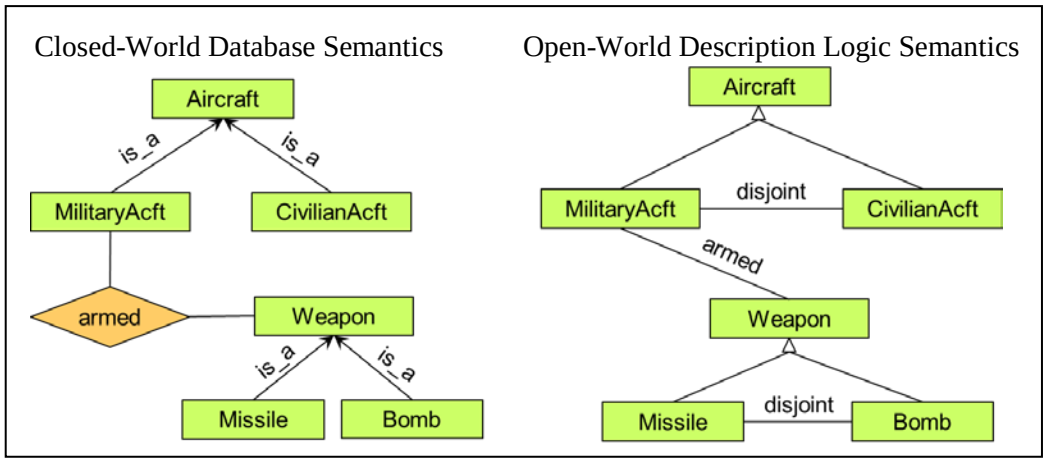


Figure 15. Closed-world (database) versus open-world (description logic) semantics

Similar relationships are depicted in the ontology diagram depicted on the right. Specifically, "MilitaryAcft" and "CivilianAcft" are depicted as (disjoint) subclasses of "Aircraft", and "Missile" and "Bomb" are depicted as (disjoint) subclasses of "Weapon", and that an entity of the class "MilitaryAcft" can have an "armed" property associating it with entities of the "Weapon" class. With open-world semantics, however, the KB does not need to include any information about "FA-18E_172396" prior to adding the assertion "armed(FA-18E_172396, AIM-120X)". Further, after inclusion of this assertion, the KB will also entail the following axioms whether they are explicitly asserted or not: "MilitaryAcft(FA-18E_172396)", "Aircraft(FA-18E_172396)", "¬CivilianAcft(FA-18E_172396)", and "Weapon(AIM-120X)".

In this case, with a single explicit assertion, we have implicitly increased the KB by five assertions whose entailment can be demonstrated through the reasoning techniques to be discussed. Note, however, that this assertion does not provide a basis for concluding that "AIM-120X" is a member of either the "Missile" or "Bomb" class (both axioms are satisfiable by the updated KB, but neither is entailed).

b. Reduction of Reasoning to Subsumption or Satisfiability

As discussed previously, subsumption and satisfiability are among the most basic DL reasoning problems. Extrapolating the definition from DLs to ontologies, subsumption can be defined as a relationship between two classes, A and B, where every individual of class A is also a member of class B (in this instance, B subsumes A). Using the OWL term for this relationship, A is a subclass of B. The DL definition of satisfiability can be applied to ontologies in similar fashion—a class is satisfiable if it is possible for an individual of the class to exist without violating the rules of the ontology.

Table 8 summarized the reduction of the key DL reasoning tasks to either subsumption or satisfiability as depicted in (Baader, et al., 08). The implication of these reductions is that if either satisfiability or subsumption can be computationally solved, then all of the problems reduceable to that task are also solvable. Stated differently, reasoning about equivalence, subsumption, satisfiability, and disjointness can all be accomplished by reasoning about either subsumption or satisfiability. In addition, TBox classification is determined through subsumption or satisfiability reasoning as well. In

practice, this reduction of standard reasoning tasks to subsumption and satisfiability also provides the basis for reasoning with ABox axioms.

Reasoning Task	Reduction to Subsumption	Reduction to Satisfiability
Equivalence	$C \equiv D$ iff $(C \sqsubseteq D)$ and $(D \sqsubseteq C)$	$C \equiv D$ iff $(C \sqcap \neg D)$ and $(D \sqcap \neg C)$ are both unsatisfiable
Subsumption	-	$C \sqsubseteq D$ iff $(C \sqcap \neg D)$ is unsatisfiable
Satisfiability	C is satisfiable iff $C \not\equiv \bot$	-
Disjointedness	C and D are disjoint iff $(C \cap D) \sqsubseteq \bot$	C and D are disjoint iff $(C \sqcap D)$ is unsatisfiable

Table 8. Reduction of standard TBox reasoning tasks to subsumption or satisfiability

Consistency checking (i.e., testing the ABox and TBox together for contradictions) can be accomplished by testing the ABox as a whole for satisfiability—the ABox is consistent with the TBox if none of the ABox axioms introduce a contradiction. Notice that consistency does not require that all TBox concepts be satisfiable, so it is possible for a consistent knowledge base to contain TBox concepts that are unsatisfiable (a TBox of this sort would be considered incoherent but satisfiable).

Entailment, or instance checking, determines whether or not a particular assertion is true for every model interpretation of the KB (regardless of whether or not it is explicitly asserted with an ABox axiom). Entailment of concept for a particular individual (i.e., $\mathcal{A} \vDash C(x)$) is easily checked by testing the satisfiability of $\neg C(x)$. If $\neg C(x)$ is unsatisfiable, then the ABox entails C(X).

Retrieval of all named individuals entailed by a concept can be accomplished by testing for entailment of each named KB individual; however, for even moderately complex knowledge bases, computational complexity makes this approach impractical (Rudolph, 11). Optimization can often be achieved, however, based on the structure of the ontology (e.g., subsumption preordering; disjoint, transitive, and reflexive relationships, etc.) (Ortiz, 10). In addition, in many cases the underlying data storage can facilitate the process by retrieving or eliminating multiple named entities at once (e.g., relational databases or XML documents to which XPath queries can be applied).

The worst-case scenario for reasoning about realization for an individual requires entailment testing of the individual for each concept in the set, S, of concepts being checked followed by subsumption testing for each of the concepts entailing the individual. If a previously determined TBox classification is available, however, realization testing only requires entailment tests of increasingly specialized concepts of S according to the preorder of the TBox.

c. Conjunctive Query Answering

Conjunctive Query (CQ) and Unions of Conjunctive Query (UCQ) answering are the most computationally expensive DL reasoning tasks. These tasks require retrieval-like reasoning for complex combinations of concepts and roles that can be comprised of multiple concepts and roles containing multiple variables. CQs and UCQs are commonly expressed as first-order logic or DL expressions or SPARQL queries.

A conjunctive query is formally defined by Equation 10, where v is a set of non-distinguished variables for which only existence is being queried, x is a set of answer variables for which the actual values are queried, and φ is a set of KB concepts and roles utilizing only elements of x and v. The answer to the query, q, is the set of answer variable tuples corresponding to query matches.

$$q = \exists v.\varphi(x, v) \qquad \text{(Eq. 10)}$$

As an example, consider the example query of Figure 16 which utilizes a FOL expression to request retrieval of all individuals, x_1, where individuals v_2 and v_3 exist such that for specific individuals, x_1, v_2, and v_3, where "Contact(x_1)", "reportedBy(v_1, v_2)", "mission(v_2, "DCA")", "tasks(v_3, v_2)" and "AirTaskingOrder(v_3)" are all entailed. In this example, x_1 is the only answer variable, so the result of the query will be the set of all contacts that were reported by a unit that was tasked by an air tasking order to conduct a "DCA" mission.

Although typically specified using FOL, a DL, or a query language (e.g., SPARQL), it is possible to express a CQ as a directed graph with nodes and arcs representing the query's concepts and roles, respectively, as illustrated in Figure 16.

Viewed in this manner, answering a query amounts to finding a homomorphic mapping from the query graph to subgraphs of KB (Ortiz and Šimkus, 12).

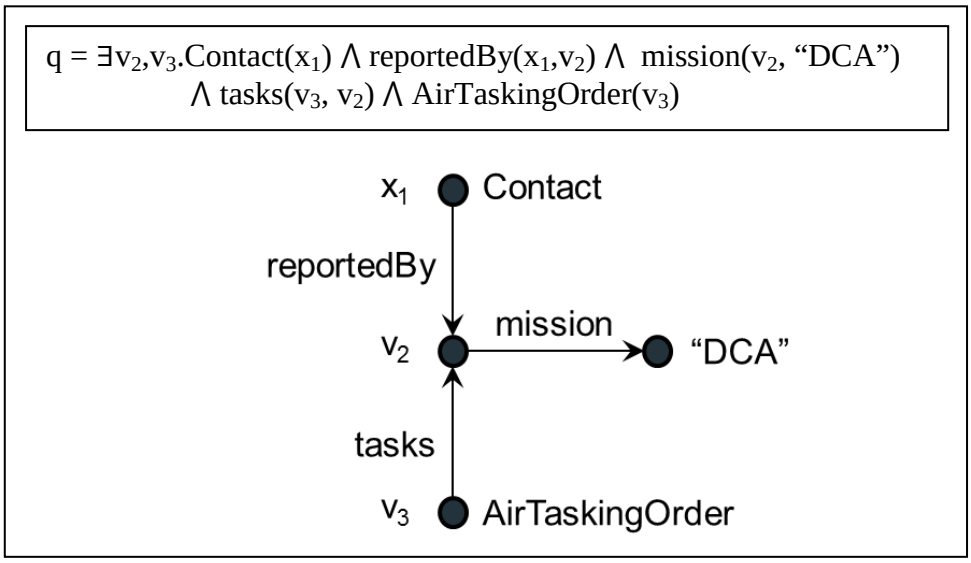


Figure 16. An example Conjunctive Query expressed with first-order logic and the associated query graph

Queries that contain no answer variables calculate a Boolean response based on the KB's entailment of any specific (i.e., bound) instances of the non-distinguished variables. Queries that contain one or more answer variables are referred to as answer queries and return tuples for all specific answer-variable bindings that satisfy the query.

In the most straightforward implementation, CQ and UCQ reasoning can be implemented by repeated testing of bindings of variables to ABox individuals and testing for entailment. This requires n^m entailment tests where n is the number of ABox individuals and m is the number of query variables. Fortunately, optimizations taking advantage of TBox-defined relationships and query structure can significantly reduce the number of required checks (Ortiz, 10).

The complexity and potential optimizations for CQ and UCQ reasoning are comparable to those of concept retrieval. Both problems have at least 2-EXPTIME-Complete complexity for many expressive DLs (Ortiz, 10). However, computational requirements for concept retrieval grow linearly with the number of named individuals in

the ABox while requirements for CQ and UCQ answering grows exponentially based on the number of named individuals in the ABox and the number of variables in the query (3-EXPTIME-Complete). Optimizations can eliminate large sections of the potential search space to make queries tractable, but CQ and UCQ remains an area of significant research interest.

d. Decidability, Complexity, and Soundness, Completeness

A few issues must be considered when assessing the suitability of a reasoning algorithm for a particular problem. First, the problem must be decidable (i.e., an algorithm must exist that will deterministically solve the problem). Although seemingly a trivial concern, subsumption and satisfiability are potentially undecidable with many expressive DLs. In fact, decidability is an open question for many potential reasoning tasks with OWL 2 (W3C, 12).

The second issue is complexity; that is, how many computational steps are required by the algorithm. Early DL reasoning algorithms operated in polynomial time; however, these algorithms proved unsuitable for expressive DLs (Baader, et al., 07). The ability to reason in polynomial time was the primary motivation behind the OWL 2 EL, RL, and QL profiles (Krötzch, 12). Reasoning with OWL 2 Full ontologies typically requires algorithms with exponential complexity (W3C, 12).

Soundness and completeness have to do with the reasoning algorithm itself. An algorithm is sound if and only if every solution that it finds is correct, and an algorithm is complete if and only if it is guaranteed to find a solution if one exists (Baader, et al., 07).

Given these requirements, tableau algorithms have been the most frequently utilized tools for ontology reasoning for production and research systems (Baader and Sattler, 01 and Rudolph, 11).

2. Reasoning Algorithms

a. Tableau Algorithms

The first example of a tableau algorithm was described in (Schmidt-Schaub and Smolka, 91) and has provided the basis for all subsequent tableau algorithms for expressive DLs. The basic tableau algorithm is useful for reasoning about satisfiability and consistency with ontologies of the $\mathcal{ALC}$ family of DLs. The algorithm

works by attempting to construct an interpretation that satisfies all of the ABox concepts and can be used to test the satisfiability of one or more concepts. The basic tableau algorithm for determining the simultaneous satisfiability of one or more $\mathcal{ALC}$ concepts is depicted in Figure 17 and utilizes the substitution transformation table of Table 9.

Algorithm:

1. Generate $\mathcal{A}_0$ as an ABox with all test axioms in Negation Normal Form
2. $S = \{ \mathcal{A}_0 \}$
3. Repeat until either termination conditions hold
   1. Choose one concept from $\mathcal{A}i$ for which a substitution table condition applies
   2. Perform the substitution table transformation to generate $\mathcal{A}_i'$ (and $\mathcal{A}_i''$)
   3. $S = (S - \mathcal{A}_i) \cup \{ \mathcal{A}_i', \mathcal{A}_i'' \}$

Termination conditions:

$\mathcal{A}_0$ is unsatisfiable: All $\mathcal{A}_i$ are closed (contain contradictions)

$\mathcal{A}_0$ is satisfiable: Any $\mathcal{A}_i$ is open (contains no contradictions) and complete (no applicable substitution table transformations)

Figure 17. Basic tableau algorithm for satisfiability testing of $\mathcal{ALC}$ axioms

ABox Axiom Condition	Transformation Action
$\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$ but not $C_1(x)$ and $C_2(x)$	$\mathcal{A}' := \mathcal{A} \cup \{ C_1(x), C_2(x) \}$
$\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$ but neither $C_1(x)$ nor $C_2(x)$	$\mathcal{A}' := \mathcal{A} \cup \{ C_1(x) \}$ $\mathcal{A}'' := \mathcal{A} \cup \{ C_2(x) \}$
$\mathcal{A}$ contains $(\exists r.C)(x)$ but not $C(z)$ and $r(x, z)$ (for any z)	$\mathcal{A}' := \mathcal{A} \cup \{ C(y), r(x, y) \}$ where y is an individual name not occurring in $\mathcal{A}$
$\mathcal{A}$ contains $(\forall r.C)(x)$ and $r(x, y)$ but not $C(y)$	$\mathcal{A}' := \mathcal{A} \cup \{ C(y) \}$

Table 9. ABox transformations for the $\mathcal{ALC}$ tableau algorithm

The first step of the algorithm is to develop a set of candidate ABoxes containing a single element. The initial candidate, $\mathcal{A}_0$, contains all axioms being tested

expressed in Negation Normal Form (NNF). NNF requires that all named concepts be atomic and that negations be applied only to atomic concepts. Conversion of axioms to NNF can be accomplished in linear time by re-expressing named complex concepts with their TBox definitions and applying de Morgan's rules (or other transformations) to negated complex concepts (Baader and Sattler, 00).

Any ABox in S that contains a contradiction (i.e., $\{ P(x), \neg P(x) \} \subseteq \mathcal{A}_i$ for some concept P and individual x) is said to be closed, while an ABox that contains no contradictions is said to be open. An ABox for which no condition from the substitution table can be applied is said to be complete.

The algorithm is complete when one of two conditions is met. If every element of S is closed, then the algorithm was unable to develop a modeling interpretation. In practice, transformation results can be tested prior to adding them to S so that closed axioms can be discounted immediately. Thus, an empty S indicates that every possible candidate ABox contained at least one contradiction. In this case, the concept(s) being tested are unsatisfiable (or the ABox is inconsistent if $\mathcal{A}_0$ contained the entire KB).

Alternatively, the algorithm is also complete when any element of S is both open and complete. In this case, the candidate ABox has been fully expanded without generating a contradiction and represents an interpretation that models the KB with the addition of the tested axioms and the concept(s) being tested have been proven satisfiable (or the ABox has been shown to be consistent if $\mathcal{A}_0$ contained the entire knowledge base).

This basic algorithm highlights two important points concerning tableau algorithms. First, the transformation associated with existential quantification generates two ABoxes. Thus, the algorithm can be viewed as generating trees with each action being applied to a leaf node to generate one or two children. The depth of the tree is bounded by the size of $\mathcal{A}_0$ and the branching factor is bounded by the number of existential quantifications. At any point in the algorithm's execution, the tree's leaves are represented by the ABoxes contained in S. The second observation is that the action

associated with a value restriction condition instantiates a new hypothetical (anonymous) individual, which is allowed under the open-world semantics.

While the $\mathcal{ALC}$ DL with which this particular algorithm works is far less expressive than OWL, it can be extended in a relatively straightforward way to work with more expressive DLs of the $\mathcal{ALC}$ family, including OWL's $\mathcal{SROIQ}$ with rules similar to those of Table 10.

ABox Axiom Condition	Transformation Action
$\mathcal{A}$ contains $(\geq nr)(x)$ but no names $z_1,\ldots,z_n$ where $r(x, z_i)$ $(1 \leq I \leq n)$ and $z_i \neq z_j$ $(1 \leq i < j \leq n)$	$A' := A \cup \{ r(x, y_i) \mid 1 \leq I \leq n \} \cup \{ y_i \neq y_j \mid 1 \leq i < j \leq n \}$
$\mathcal{A}$ contains $(\leq nr)(x)$ and $r(x, y_i)$ for all $i = 1$ to $n+1$ but not $y_i \neq y_j$ for some i and j $(1 \leq i < j \leq n+1)$	For y_i and y_j $(1 \leq i < j \leq n)$ where $y_i \neq y_j$ is not in A, generate $A_{i,j}$ by replacing each occurrence of y_i in A with y_j
$\mathcal{A}$ contains $(\leq nr.C)(x)$ and $r(x, y)$ but neither $C(y)$ nor $\neg C(y)$	$A' := A \cup \{ C(y) \}$ $A'' := A \cup \{ \neg C(y) \}$
$\mathcal{A}$ contains $(\exists r.C)(x)$ but not $C(z)$ and $s(x, z)$ (for any z, s is a sub-role of r)	$A' := A \cup \{ C(y), r(x, y) \}$ where y is an individual name not occurring in A
$\mathcal{A}$ contains $(\forall r.C)(x)$ and $s(x, y)$ (where s is a sub-role of r) but not $C(y)$	$A' := A \cup \{ C(y) \}$
$\mathcal{A}$ contains $(\forall s.C)(x)$ and $r(x, y)$ and r is a transitive sub-role of s	$A' := A \cup \{ (\forall r.C)(y) \}$
$\mathcal{A}$ contains $\bar{s}(x, y)$ and $(\forall r.C)(y)$	$A' := A \cup \{ C(x) \}$

Table 10. Tableau algorithm ABox transformations for $\mathcal{ALC}$ extensions

The first two rules provide for unqualified cardinality restrictions (the $\mathcal{ALCN}$ DL). The first rule accounts for greater-than rules by adding enough unique role instances to $\mathcal{A}$ to meet the required minimum. The second accounts for less-than rules by combining role instances that have not been explicitly declared unequal (i.e., replace "$r(x, y_1)$" and "$r(x, y_2)$" with a single occurrence of "$r(x, y_1)$" unless A contains an axiom asserting "$y_1 \neq y_2$"). The method of checking $\mathcal{A}$ for closure must be augmented slightly to account for these new rules as well.

The third rule provides for qualified cardinality restrictions (the $\mathcal{ALCQ}$ DL) by adding "C(y)" to $\mathcal{A}'$ and "¬C(y)" to $\mathcal{A}''$ for all y in the qualified role with x that are not already identified in $\mathcal{A}$ as either C or ¬C.

The fourth and fifth roles provide for role hierarchies (role/sub-role relationships) by modifying the existential quantification and value restriction rules of the original algorithm.

Finally, the sixth and seventh rules provide for transitivity and inverse roles.

In combination, these rules provide a basis for algorithmically reasoning with ontologies with $\mathcal{SHIQ}$ DL semantics and can be extended further to reason with $\mathcal{SHOIQ}$ ontologies expressible with OWL.

b. Automata-Based Reasoning

Automata-based reasoning for DLs has been a topic of significant recent research. Automata-based approaches have limited implementation examples to date because of the requirement to generate a potentially exponentially large automaton (Calvanese, et al., 11). Nevertheless, they have characteristics that may provide for improved performance for reasoning tasks for which tableau algorithms are demonstrably inefficient such as retrieval and conjunctive query answering (Ortiz, 10).

Automata-based techniques are similar to tableau algorithms in that they conduct queries by verifying (or refuting) the existence of modeling interpretations that satisfy the reasoning task requirements (Rudolph, 11).

Automata-based algorithms rely on KB forest interpretations that are constructed in accordance with the rules of Figure18. This type of interpretation serves two purposes: 1) it limits counter-example search area (which is the mechanism used by both tableau and automata-based algorithms), and 2) automata techniques have been utilized successfully with infinite tree structures on problems closely related to DL reasoning (Ortiz, 10).

The second component of automata-based reasoning systems is an alternating tree automata (ATA) that accepts or rejects KB trees (or more recently, a whole forest) (Ortiz and Šimkus, 12). An ATA is a nondeterministic finite automaton

that accepts trees as input. Nondeterminism allows the automata to resolve unions, intersections, cardinality restrictions, or other operations with potentially disjoint or parallel realizations. In these instances, the algorithm branches to test all possible or necessary realizations (Glimm, et al., 08).

All automata-based algorithms follow the same general procedure described in Figure 18. First an ATA, A_1, is developed that will accept forest interpretations that model the KB. Second, a query-specific automaton, A_2, is developed that will accept trees containing query matches. The intersection of A_1 and the complement of A_2 provides an automaton that accepts counter-models to query entailment (i.e., it accepts trees that model KB for which the query is false). Because query entailment requires that the query be true for every model of KB, the set of trees accepted by $A_1 \cap \neg A_2$ will be empty if the query is entailed (Calvanese, et al., 09 and Calvanese, et al., 11).

<table>
<tr>
<td>Knowledge-Base Forest Interpretation
Assign each $a \in \mathcal{A}$ as a root node
Nodes, w and w′ are part of the interpretation if
w is a child of w′
w′ is a child of w
w or w′ is a root node, or
$(w, w') \in r^I$ (including transitivity)</td>
<td>Alternating Tree Automaton (ATA)
$A = < \Sigma, Q, q_0, k, \delta >$
Σ is the input alphabet
Q is a set of states
q_0 is the initial state
k is the input tree branching factor
δ is a transition function</td>
</tr>
<tr>
<td colspan="2">Basic Automata-Based Description Logic Reasoning Algorithm
Develop distinct ATAs for $\mathcal{KB}$ and the query, q
Develop an ATA, A_1, that accepts forest interpretations modeling $\mathcal{KB}$
Develop an ATA, A_2, that accepts trees that satisfy the query, q
$A_1 \cap \neg A_2$ is an ATA that accepts countermodels to query entailment
Query entailment is reduced to an automaton emptiness test
$\mathcal{KB} \models q$ if and only if $L(A_1 \cap \neg A_2) = \emptyset$</td>
</tr>
</table>

Figure 18. Automata-based DL reasoning about query entailment

Automata-based algorithms have been utilized for many expressive DLs approaching the expressiveness of OWL 2. (Calvanese, et al., 09) and (Calvanese, et al., 11) propose automata-based algorithms for reasoning with the $\mathcal{SRIQ}$, $\mathcal{SROQ}$, and $\mathcal{SROI}$ DLs. Algorithms for CQ reasoning with DLs more expressive than these are not available at present, and the question of whether or not CQ reasoning with full $\mathcal{SHOIQ}$ DL of OWL 1 or $\mathcal{SROIQ}$ of OWL 2 is computationally decidable is currently an open question (Rudolph, 11).

c. Resolution-Based Reasoning

Tableau algorithms and automata algorithms both reason about satisfiability by attempting to develop a model for the expression being tested (i.e., if a model exists, then the expression must be satisfiable). Methods of this sort utilize model-theoretic reasoning (Rudolph, 11). Resolution, on the other hand, uses a proof-theoretic approach where the axioms of KB are converted to Clause Normalized Form (CNF) FOL clauses to which resolution calculus rules are applied. A CNF FOL representation that is equivalent to a DL KB will have the following characteristics:

- DL axiom elements are replaced with substitutions from Table 11 (and others not depicted)
- Negations are applied only at the atom level
- Existentially quantified variables are Skolemized (e.g., "∃armed.Sidewinder" will be replaced by "armed(x, f(x)) ∧ Sidewinder(f(x))"
- All other variables are implicitly universally quantified
- Clauses are manipulated to eliminate embedded conjunctions
- Conjunctive clauses are separated into distinct clauses

Once the KB has been converted to CNF FOL clauses, the query is negated, converted to CNF clauses, and notionally added to the KB. Unification rules are then iteratively applied until an inconsistency is discovered or all possible unifications have been tried. Unifications are conducted by finding two clauses with complimentary atoms, at least one of which contains an unbound variable. For example, the clauses "¬A(x) ∨ B(z) ∨ s(x, z)" and "C(a) ∨ r(b, c) ∨ ¬s(b, 5)" can be unified because the first

contains “s(x, z)” and the second contains “¬s(b, 5)”. The unification rule relies on a unification pairs of { b/x, z/5 } as to yield the following addition to the KB: “¬A(b) ⋁ B(5) ⋁ C(a) ⋁ r(b, c)”. If an inconsistency is uncovered (i.e., KB includes clauses for both A(x) and ¬A(x)), then the original query, q, has been shown to be entailed (since KB ∪ { ¬q } is unsatisfiable).

Axiom Term	First Order Logic Clause
⊤ or ⊥	True or False
C or r	C(x) or r(x, y)
C(a) or r(a, b)	C(a) or r(a, b)
¬C or ¬r	¬C(x) or ¬r(x, y)
C ⊑ D	C(x) ⋁ ¬D(x)
r ⊑ s	r(x, y) ⋁ ¬s(x, y)
C ⊓ D	C(x) D(x)
C ⊔ D	C(x) ⋁ D(x)
∃r.B	r(x, f(x)) B(f(x))
∀r.B	¬r(x, y) ⋁ B(y)

Table 11. First Order Logic substitutions for DL axioms

Resolution-based algorithms have been described for DLs up to $\mathcal{SHOIQ}$ with specific restrictions applied to ensure termination and decidability (Motik and Sattler, 06 and Kazakov and Motik, 08).

3. Inferencing with Inductive Rules

As has been covered previously, inductive rules that extend a DL can allow for the expression of relationships that cannot be defined solely with DL. Trigger rules consisting of an antecedent and a consequent can be used to add new facts to the database—if the antecedent of the rule is satisfied, then the consequent of the rule can be added to the KB. Alternatively, rules can be viewed as augmenting satisfaction- and subsumption-based reasoning. If the TBox includes a set of trigger rules, then all ABox axioms that are entailed by the ontology are implied by the rules and other TBox axioms

even if they are not explicitly stated as ABox axioms. Reasoning about specific trigger rule goals, then, can be viewed a different form of entailment testing.

Two general methods are used for rule-based inferencing. Forward chaining is a data driven approach. Forward chaining algorithms begin by identifying all rule antecedents that are matched by existing KB axioms. This set of applicable rules is referred to as the conflict set. Selected rules from the conflict set are fired to assert new axioms, and the process repeats until a specified goal is achieved, the rule matcher returns an empty conflict set, or a predetermined number of algorithm iterations has been executed. Forward chaining is most applicable when the intent is to build the knowledge base out as much as possible rather than to determine whether or not a specific goal is entailed by the KB (although conflict resolution optimization can make forward chaining suitable for reasoning about specific goal entailment).

Backward chaining is a goal-driven approach to reasoning. These algorithms begin with a specific goal and attempt to find an inference chain that will support the addition of the goal to the KB. The initial conflict set for a backward chaining algorithm, then, contains all rules for which the goal is the consequent. A goal from the conflict set is selected and used as a sub-goal for a recursive call to the algorithm. The recursive base case is reached when the goal is contained in the KB (success) or is not the consequent of any rules (failure). Backward chaining can be naïvely implemented with a simple depth-first or breadth-first strategy, but can be more efficiently implemented with directed search strategies such as A* search, means-ends analysis, or GraphPlan (Russell and Norvig, 10). Not surprisingly, backward chaining algorithms are useful when the intent is to determine whether or not a specific goal is entailed by the KB. From this standpoint, backward chaining is analogous to entailment checks discussed previously.

A number of issues might be apparent in backward chaining. First is the question of antecedent matching tractability. Taken to the extreme, matching rule antecedents potentially requires conjunctive query answering for every rule in the KB. In practice, this is unrealistic, so rule matches are often based on axioms that are explicitly included in the KB. Additionally, the composition of rule antecedents can be restricted to improve matching performance and maintain decidability (Horrocks, et al., 04).

The second issue regards conflict resolution, which is of particular importance with forward chaining algorithms (but is relevant with backward chaining as well). If the KB is large, the conflict set is likely to be large as well, making the algorithm's performance highly dependent upon the order in which new axioms are added to the KB. The selection strategy can be hard-coded into the reasoner, or it can be included as part of the model.

Decidability is a function of the DL itself and also of the language used to specify trigger rules. Trigger rules are generally restricted Horn clauses with additional restrictions to the antecedent to make the rule matching process tractable. Termination of both forward and backward chaining algorithms is guaranteed by the finite size of the KB (Baader, et al., 07).

Finally, consistency of the KB is potentially more problematic if trigger rules are included. The consistency check algorithms that have been discussed thus far do not account for trigger rules, so the possibility that a trigger rule will make an assertion that compromises the ontology's consistency cannot be discounted.

B. COMPARISON OF AVAILABLE SEMANTIC WEB REASONERS

Both commercial and open source reasoners are available for use with OWL ontologies. A few of the more popular and more capable ones are included in this comparison. A tabular comparison is provided in Table 12.

- Pellet (Sirin, et al., 07), a Java-based, open source reasoner for OWL ontologies, was the first reasoner to fully implement the full OWL 1 DL capability (the $\mathcal{SHOIN}$ DL) and has been extended to support OWL 2 features ($\mathcal{SROIQ}$ DL). It uses an optimized tableau algorithm for most reasoning tasks and an optimized "rolling up" technique (which, like tableau and automata algorithms, leverages an ontology's tree model property) for conjunctive queries with non-distinguished variables. In addition, Pellet includes support for multi-ontology reasoning and non-monotonic reasoning.
- RacerPro (Haarslev, et al., 12) is a commercial product of Racer Systems GmbH & Co. KG for reasoning with the $\mathcal{SHIQ}$ subset of OWL 2. It uses an optimized tableau algorithm for reasoning. A robust proprietary language (nRQL) is used for specifying conjunctive queries and inductive rules (including facilities for defining when rules fire and under what circumstances they are active), and RacerPro also supports SPARQL queries and SWRL

rules. In addition, RacerPro provides a number of interesting additions such as TBox and ABox retractions (i.e., non-monotonicity).

- FaCT++ (Tsarkov and Horrocks, 06) is an open source reasoner providing full support for OWL 1 DL and partial support for OWL 2. FaCT++ uses an optimized tableau algorithm for reasoning, but does not support rules and only supports a restricted set of conjunctive queries.
- The Scaleable Highly Expressive Reasoner (SHER) (Dolby, et al., 09) is a commercial product of IBM developerWorks® that is built upon the functionality of the open source Pellet reasoner. SHER is specifically designed to work with very large knowledge bases. Efficiency is gained by reasoning with an in-memory summary of the complete ABox, through the use of polynomial-time algorithms for subsets of the full $\mathcal{SHIN}$ DL (e.g., $\mathcal{EL}$+), and applying fast and sound (but not necessarily complete) reasoners to find obvious solutions quickly.
- Jena (Apache, 13) is an open source Semantic Web framework with a Java Application Programming Interface (API) for working with RDF graphs. Although intended for RDF, Jena does provide support for OWL. It does not fully support any specific DLs, however, and as a result, provides only limited reasoning capabilities. It does provide rule support, but utilizes its own format and does not allow rules to be specified with SWRL.
- The KAON2 reasoner (Motik, 08) and its commercial counterpart, Semafora Systems' OntoBroker, are unique among the reasoners discussed here in that it utilizes a resolution-based algorithm for reasoning. KION2 uses a client-server system for maintaining ontologies and supports conjunctive queries specified in SPARQL and inductive rules specified in SWRL. KAON2 also provides a Java API for program management, manipulation, and reasoning with ontologies.

Reasoner	DL	Algorithm	Entailment	Consistency	Rules	CQs
***Pellet**	$\mathcal{SROIQ}$	Tableau	Yes	Yes	Yes	Yes
****RacerPro**	$\mathcal{SHIQ}$	Tableau	Yes	Yes	Yes	Yes
***FaCT++**	$\mathcal{SROIQ}$	Tableau	Yes	Yes	No	Partial
*****SHER**	$\mathcal{SHIN}$	Rules	Yes	Yes	Yes	Yes
***Jena**	None	Rules	Partial	Partial	Yes	No
****KAON2**	$\mathcal{SHIQ}$	Resolution	Yes	No	Yes	Yes
*Open source product, **Free closed-source product, ***Commercial product						

Table 12. Comparison of commercial and open source DL reasoners

C. NON-STANDARD DESCRIPTION LOGIC REASONING TASKS

With the exception of inductive rules, all of the reasoning tasks discussed thus far can be reduced to satisfaction checking. These *standard reasoning tasks* are deductive in nature and draw logical conclusions from the KB itself. Although inductive in nature, rule systems also provide a mechanism for drawing logical conclusions from the knowledge base. In all of these cases, the reasoning conclusions are guaranteed to be entailed by KB. On the other hand, it is sometimes desirable to make inferences from a KB base that are not fully entailed by the axioms contained in the KB.

Induction is a process of drawing generalized conclusions from a KB's assertional data or drawing conclusions about individuals or groups represented in KB that are not fully entailed. Induction relies heavily on concepts of machine learning and data mining that will be discussed shortly. This sort of reasoning is useful when decisions or conclusions need to be based on a preponderance of the evidence rather than on conclusive evidence. Historically, Inductive Logic Programming (ILP) (Muggleton and Raedt, 94) has proven particularly applicable in the area of induction reasoning with ontologies (Rudolph, 11).

Abduction, on the other hand, attempts to identify missing premises that if present would result in the entailment of desirable (or presumed) axioms. Stated differently, if an axiom, α is not entailed by KB, what additional axioms, KB', would result in α's entailment were they added to KB. Abductive reasoning services are useful when a desirable or suspected outcome is not entailed, and one wants to identify the missing information. Abduction is also an intuitive application of ILP (Muggleton and Raedt, 94).

It is important to note that, unlike standard reasoning tasks, induction and abduction algorithms are not truth preserving. That is, assertions may be added to KB that turn out to be false as more information is gathered.

THIS PAGE INTENTIONALLY LEFT BLANK

V. MACHINE LEARNING AND THE SEMANTIC WEB

A. OVERVIEW

The most prominent areas of machine learning research relating to the Semantic Web concern the use of machine learning techniques to develop and maintain ontologies. Machine learning was identified early on as a means of facilitating the growth and development of the Semantic Web, with an early W3C white paper proposing five specific applications (Maedche, 01). These can be roughly divided into the use of machine learning to build Semantic Web content from existing web data and the use of machine learning to improve and maintain existing Semantic Web content. Ontology extraction and Metadata extraction fall into the first category, while ontology merging, ontology maintenance, and application management fall into the second.

Ontology extraction from existing web data requires the analysis of existing structured and unstructured data to identify relationships and concepts. Data can range from completely unstructured material such as written documents, images, and streaming data to highly structured data stores such as relational databases, taxonomies, and dictionaries.

Extraction of relational metadata from existing web data involves the identification of characteristics of individuals and relationships between individuals. Metadata extraction might be viewed as a preliminary step in automated ontology development in that characteristics and relationships identified in this stage can be leveraged later.

Merging and mapping ontologies involves identifying common concepts, roles, and individuals. Although ontology matching might be viewed as simpler than developing an ontology from scratch, it is complicated by the fact that similar terms may be defined differently and common concepts and relationships can have overlapping but not identical meanings. As pointed out previously, current automated ontology merging approaches provide recommendations, but do not typically merge concepts, roles, or individuals without the concurrence of a human supervisor.

Maintaining ontologies by analyzing instance data involves the development of new TBox axioms from instance data contained in the ABox. Commonly referred to as

ABox mining, this operation typically involves analyzing RDF statements to derive taxonomical classes and their relationships. Metadata extraction and ABox mining together can be used as sequential steps in the larger ontology extraction process.

Finally, the use of machine learning for application maintenance has been proposed as a means of improving Semantic Web services through analysis of user activity.

Despite its importance to the evolution of the Semantic Web, automated generation of Semantic Web content through machine learning is not as important to data fusion and correlation. These tasks might, however, benefit from the application of machine learning techniques. In particular, the inductive reasoning task is the assertion of new ABox axioms to capture presumed or predicted information. The process amounts to drawing the most likely conclusions based on the available data, and is clearly analogous to traditional machine learning.

When compared to the use of machine learning to build the Semantic Web, examples of existing Semantic Web data use in support of machine learning is less ubiquitous. This is in part due to the fact that most of the web is still not yet semantically described. Nevertheless, Semantic Web content has characteristics that are well-suited to its use in machine learning, and mining Semantic Web content is an active area of research (Tresp, et al., 08). Specifically, semantic description of web data removes ambiguity, imposes structure, and captures background information. All of these can be used by machine learning techniques that rely on interpretations of web content and organization of individual entities.

Semantic Web content can be effectively applied to most machine learning techniques; however, there are a few techniques that have proven particularly relevant.

B. INDUCTIVE LOGIC PROGRAMMING

ILP combines aspects of propositional logic, inductive learning, and logic programming to derive inference rules (Tresp, et al., 08). The idea is to apply propositional calculus to a KB in a way that derives new inductive rules from existing rules (the TBox) and evidence (the ABox) so that the rules fully explain the evidence.

The general premise of all inductive inference can be stated as follows: given background knowledge, B, and evidence, E, where E does not contradict B (prior satisfiability) but B does not fully explain E (prior necessity), inductive inference attempts to find a hypothesis, H, such that adding the hypothesis to the background knowledge fully explains the evidence (posterior sufficiency) and maintains satisfiability of the knowledge base (posterior satisfiability) (Muggleton and Raedt, 94). This can be expressed in terms of a KB consisting of a TBox, $\mathcal{T}$, and an ABox, $\mathcal{A}$, where the evidence, $\mathcal{E}$, is a subset of the ABox ($\mathcal{E} \subseteq \mathcal{A}$) with the following expressions:

- Prior satisfiability: $\mathcal{T} \cup \mathcal{E} \not\models \bot$
- Prior necessity: $\mathcal{T} \not\models \mathcal{E}$
- Posterior satisfiability: $\mathcal{T} \cup \mathcal{E} \cup \mathcal{H} \not\models \bot$
- Posterior sufficiency: $\mathcal{T} \cup \mathcal{H} \models \mathcal{E}$

The ILP algorithm works by iteratively applying transformation rules from a rule set, R, to conjunctions of clauses to make them more generalized or specialized. A conjunction of clauses, G, is said to be more general than a conjunction of clauses, S, if and only if G entails S. Conversely, G is said to be more specialized than S if and only if S entails G. Rules are considered either deductive or inductive based on whether they perform a specialization or generalization role, respectively (Muggleton and Raedt, 94). As an example, consider the following rule for absorption (A and B are unbound variables for clauses, and p and q are unbound variables for atoms):

$$\text{Absorption: } \frac{p \leftarrow A \wedge B \qquad q \leftarrow A}{p \leftarrow q \wedge B \qquad q \leftarrow A}$$

This is an inductive rule that can be applied to any pair of clauses of the form "p ← q ⋀ B" and "q ← A" to yield two new clauses with forms "p ← A ⋀ B" and "q ← A". The reverse of the rule can applied deductively, to clauses of the form "p ← A ⋀ B" and "q ← A" to generate new clauses of the form "p ← q ⋀ B" and "q ← A" (this is essentially resolution).

It must be noted that inductive rules are not logically sound, meaning that they are a mechanism through which the truthfulness of a KB can be compromised. Consider the preceding example. If the KB contains a clause of the form "q ← C" in addition to "q ← A" (for the same q), then the antecedent of the added clause, "p ← q ⋀ B", can be satisfied in cases where the antecedent of original clause, "p ← A ⋀ B", is not (e.g., "¬A ⋀ C" is entailed).

The basic ILP algorithm as depicted in Figure19 works by maintaining a queue of candidate hypotheses, QH. At each iteration of the algorithm, a single hypothesis is removed from QH and a set of applicable rules are chosen from R. The rules are applied (either exhaustively or by some other criterion) to H to produce a new set of hypotheses, H_1 through H_n. Promising hypotheses are added to QH and the process is repeated until a specified completion criterion is reached. Generated hypotheses are essentially children of previous hypotheses, so the ILP algorithm can be viewed as an extension of standard DL reasoning algorithms that leverage the tree-model property of ontologies to derive models for KB.

```
QH := initialize
repeat
   H = dequeue( QH )
   choose( R_H ⊆ R ) where R_H = { ∀r_k | r_k to be applied to H }
   apply rules r_1, …, r_k to H to yield H_1, …, H_n
   for each i = 1 to n
      enqueue( H_i, QH )
   prune( QH )
until stop-criterion( QH ) satisfied
```

Figure 19. The Inductive Logic Programming algorithm (Muggleton and Raedt, 94)

The algorithm contains a number of generic procedures (denoted with italics) that must be defined for the particular application. The initial QH may contain a single hypothesis containing the ground truth TBox and ABox (or subsets), or it may contain

one or more hypotheses that include desirable or suspected axioms (i.e., abductive reasoning as previously defined).

The set of inference rules, R, consists of an arbitrary set of inductive and deductive rules as described. R can contain both domain-independent rules such as the absorption example and domain-specific rules containing information directly applicable to a certain KB.

Rules are chosen based on applicability to the hypothesis being expanded and according to application-specific criteria. It is not necessarily the case that all applicable rules will be selected in this step. As an example, it is permissible to apply rules either deductively or inductively, but in most cases inductive application is preferred (deduction can be accomplished through standard DL reasoning techniques). Additionally, exhaustive rule application will lead to exponential growth of QH, so the rule-selection heuristics must prioritize rules based on whether or not they make progress towards the desired end state and select rules for inclusion in R_H accordingly.

The makeup of QH requires consideration of similar issues. The depicted algorithm uses the standard "enqueue" and "dequeue" terminology from the computer science field to represent the operations for adding and removing hypotheses from QH. In practice one or both of these operations must account for the likelihood that particular hypotheses will lead to the best solution. Even with highly selective rule selection heuristics, traditional queue operations will result in an inefficient breadth-first search of hypotheses. Efficiency can be improved by implementing QH as a priority queue ordered according to an evaluation metric. The evaluation metric provides an assessment of each hypothesis based on its proximity to a solution and effectively implements the algorithm as a best-first search.

The algorithm's "prune" operation allows the removal of impossible hypotheses without further evaluation. This step is required because rules from R are applied irrespective of the evidence—there is no requirement that the consequent of a rule maintain ABox consistency. Because the final solution must meet the posterior satisfiability requirement, hypotheses that are inconsistent with the KB can be eliminated without further evaluation.

In addition to eliminating inconsistent hypotheses, the pruning function can be used to eliminate highly unlikely hypotheses. As discussed earlier, the application of inductive rules introduces uncertainty—the likelihood of a new hypothesis is a function of the likelihood of its parent hypothesis and the uncertainty introduced by the inductive rule. Each hypothesis, therefore, can be assessed according to its likelihood, and the unlikely ones can be eliminated without further evaluation (Muggleton and Raedt, 94). Bayesian approaches that probabilistically evaluate hypothesis likelihood based on the empirical probabilities contained in the KB and inductively deduced additions can be intuitively applied for this purpose. It might also be the case that an oracle (i.e., a user or other arbiter) can be invoked to eliminate unsuitable hypotheses.

Evidently, the stop-criterion function can be satisfied in one of two cases. Either a hypothesis has been derived that meets both the posterior satisfiability and posterior sufficiency requirements, or QH contains no more hypotheses for evaluation. In the first case, the inductively derived rules of the satisfying hypothesis can be added to the KB to "explain" the tested subset of the ABox (or they formulate the missing information in the case of abductive reasoning). An empty QH, on the other hand, indicates that the KB does not contain enough information to explain the initial hypotheses, but it is important to recognize that this does not equate to a refutation of the initial hypotheses. Additionally, computational exigencies might necessitate the imposition of other stop criteria. In most cases this will amount to restricting the algorithm to a predetermined number of iterations.

ILP is a well-researched learning mechanism that is well suited to reasoning with DLs. The most significant limiting factor is computational complexity that does not scale well and will limit its usefulness with large ontologies (Rettinger, et al., 12).

C. FEATURE-BASED STATISTICAL LEARNING

Feature-based statistical learning treats relationships in a KB as random variables where RDF triples are associated with probabilities equating to confidence levels (Trest, et al., 13). Probabilities are computed based on statistical measurements on a representative population. The population that is the subject of the algorithm is typically comprised of a set of tuples equating to a query response. For instance, a particular

learning algorithm population from a command and control system might contain tuples of the form "< contactID, contactType, contactSource >". Individual members of the population set are called statistical units.

Features of interest for each for the population include both independent (or explanatory) variables and dependent (or predicted) variables. Independent variables are those that the learning algorithm will use to derive predictions. Independent variables can include feature values that are used to define the population and any additional features. Dependent variables are those whose values the algorithm will attempt to predict. (Trest, et al., 08)

In the first two steps of the typical statistical learning algorithm (Figure 20), the KB is queried to retrieve tuples of the target population, P, and the set of all population members is sampled to yield a training set, S, of statistical units that are representative of the overall population. The query-sample sequence of these steps presumes a distributed ontology of linked data where query retrieval takes the form of a web search. With distributed source retrieval, identification of all population instances is unrealistic, but a representative sample can be obtained. It is important, however, to minimize the effects of search engine bias.

$P = query_P(\mathcal{KB})$ (population)
$S = sample(P)$ (training set)
$M = featureMatrix(S)$ (1 row per statistical unit, $s \in S$, 1 column per feature)
for each $i = 1$ to $count(rows)$
    for each $j = 1$ to $count(columns)$
        $M_{i,j} = v_{i,j}$ where $\mathcal{KB} \models feature_j(s_i, v_{i,j})$
        $M_{i,j} = N/A$ (or 0) where $\mathcal{KB} \not\models feature_j(s_i, x)$ for any x
Derive a statistical model for M
Apply model to all of $\mathcal{KB}$

Figure 20. Statistical learning algorithm for DL induction

Following generation of the training set, a feature matrix, M, is constructed where each row corresponds to a statistical unit of the sample set and each column corresponds

to a feature. Matrix entries, therefore, correspond to feature values for specific statistical units. It is often the case that statistical units do not have values for every feature. In these cases a value of "N/A" or "0" can be entered into the matrix to signify a non-value.

Following generation of matrix, M, analysis is conducted to derive a statistical model. Strictly speaking, there are numerous machine learning paradigms that can be used for this task including case-based reasoning, neural networks, genetic algorithms, and many others. It is usually the case, however, that the learning algorithm is used to generate a probabilistic Bayesian model for M. Commonly utilized models include naïve Bayes models, hierarchical Bayesian models, and Bayesian networks (Trest, 08).

When developing the statistical model of M, it is important to properly account for missing data. Feature-based statistical learning sometimes imposes closed-world semantics on M in which missing values are assumed to be false or zero. This approach does not work well with a sparse matrices (which is often the case), because treating missing values as false can unacceptably skew the results. For this reason, missing values are often ignored during statistical analysis or assigned values that minimize their impact on the model.

D. RELATIONAL MATRICES AND TENSORS

Relational matrices and tensors can be viewed as an extension of the feature-based statistical learning paradigm in that they represent characteristics of individuals as matrices of truth values. The use of relational matrices as a machine learning mechanism is well-documented, and they have proven applicable to a large class of relational learning problems (Singh, 09). Given this history, the applicability of relational matrices and tensors to learning with semantic graphs and ontologies is fairly obvious.

The general idea is that a matrix is used to represent a single ontological relationship and all potential participants. The rows and columns of the matrix represent individual entities that may participate in the relationship. Matrix cells are assigned a value of "1" if the relationship exists (in row-column order) between the entities in the row and column and "0" otherwise.

Each entity in a matrix is described by a small set of parameters that play the role of independent variables. As a rule, if an entity is present in more than one matrix, the

same parameters will be used for every matrix. The matrices themselves contain the dependent variables.

Learning is conducted by decomposing the matrix using factorization functions to generate an approximation of the original matrix. After matrix reconstruction, the contained values can be interpreted as confidence values that the relationship holds between the represented values (Rettinger, et al., 12). (Singh, 09) provides an overview of factorization techniques that can be utilized with relational matrices with emphasis on scalability.

Until recently, most applications of relational matrices to learning Semantic Web relationships focused on individual matrices (Trest, et al., 08). More recently, work has expanded to the use of 3-dimensional tensors that are comprised of layered relationship matrices (Huang, et al., 13 and Rettinger, et al., 12). These approaches have a number of potential advantages. First, the tensor can be sliced in various ways to process the contents from different perspectives. For instance, a frontal slice contains a matrix for a single relationship as previously described, while a horizontal slice describes all relationships as they relate to a single subject individual and lateral slices describe the relationships as they relate to a single predicate individual. These multiple perspectives enable the consideration of all relationships simultaneously so that their interelationships can be captured (Huang, et al., 13).

Relational matrices are among the most promising learning mechanisms for Semantic Web applications because of their scalability, which is a byproduct of the sparse nature of most tensors (Rettinger, 12).

E. ADDITIONAL SEMANTIC WEB LEARNING TECHNIQUES

ILP and feature-based learning (which includes relational matrices and tensors) are among the most important approaches to machine learning with Semantic Web content. There are other approaches, however, that have shown promise and bear mentioning.

Instance-based learning utilizes a feature vector to identify concepts applying to individuals. The algorithm works by comparing the feature vector of an individual being tested to feature vectors associated with a particular class or concept (determined through

a training process). If the comparison meets specified criteria, then the test individual is presumed to be a member of that class. The most common comparison mechanisms use distance functions that take in numerical feature vectors for two individuals and a weight vector and compute a disimilarity measure between 0 and 1 (d'Amato, et al., 08). The individuals characterized by the input feature vectors are determined to be members of the same class if the disimilarity measure is below a predetermined threshold.

Instance-based learning is well-suited to testing whether or not a concept can be applied to an individual. Not surprisingly, it is most widely used for instance checking and retrieval. Although the application scope is limited, instance-based learning is more efficient than standard reasoning algorithms, so it remains popular for these tasks (Rettinger, et al., 12).

Kernel functions are functional mappings from an input domain (individuals) to a feature space. The function effectively partitions the feature space into regions such that application to an individual will map it to the region most appropriate for its classification. Support Vector Machines are the most well-known kernel functions; however, kernel functions can be developed to support many well-known learning algorithms (Rettinger, et al., 12).

The first kernel functions applied to Semantic Web content assessed the logical structure of the individual being tested and the semantics of the primitive concepts (Fanizzi and d'Amato, 07). Logical structure kernel functions often resemble the similarity functions used with instance-based reasoning. Kernels that work with a portion of the semantic network graph associated with the individual being tested are also popular. These methods have the advantages of not requiring manual feature definition and of not being based on *a priori* assumptions about the data structure (Rettinger, et al., 12).

Relational Graphical Models (RGM) represent ontological statements with random variables and can be thought of as extensions of earlier models including Bayesian networks, Markov networks, dependency networks (Trest, et al., 08). RGMs represent all possible links in an RDF graph as nodes (vertices), where the actual existence of the link is probabilistically expressed as a binary random variable. Connections (edges) in the graph represent interdependencies between the nodes.

Probabilistic Relational Models (PRM) are a form of RGM where nodes capture the probability distribution of object attributes and links represent relationships between objects. Early PRMs required that the relationships between objects be known, but extensions have made it possible to utilize PRMs to consider cases where relationships are unknown (Getoor, et al., 07).

Markov Logic Networks (MLNs) follow the pattern described above for defining nodes and links. Probabilities are formulaically assigned based on the number of grounded inputs to the nodes (i.e., the confidence in the values upon which the relationships rely) and weights assigned to each formula (Trest, et al., 08). MLN learning involves estimating the appropriate weights for each formula.

Finally, Latent-Class RGMs attempt to incorporate hidden variables that may be present in an ontology. These hidden variables are incorporated into a Bayesian network comprised of relationships specified by the KB. Presumed latent variables are introduced for each entity as a parent of all nodes with which the entity is involved. Because the links in the underlying Bayesian network are completely specified by the KB, training with Latent-Class RGMs amounts to determining weights for nodes corresponding to latent variables (Trest, et al., 08).

The most significant drawback to RGMs is that they are not factorable into individual data points, meaning that the whole data set essentially comprises a single data point. This can significantly complicate inferencing and learning (Rettinger, et al., 12).

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The difference between information and knowledge is an important factor when considering solutions to the data fusion problem. The amount of data available to operators and decision-makers has exploded in recent years, but increased data availability has not necessarily translated into more efficient operations or better decision-making. The term "information overload" has become a cliché, but it accurately reflects the current state of affairs.

What is required are means of automatically converting the abundance of information into an abundance of knowledge. This includes mechanisms for computationally interpreting, categorizing, and correlating information to develop a knowledge base and using that knowledge base to draw conclusions, make predictions, and aid the decision-making process. This is an area to which the Semantic Web technologies described in this report can be directly applicable.

Description Logics (DL) provide the mathematical foundation for the Semantic Web. A subset of First-Order Logic, DLs use concepts and roles to describe individuals and their relationships. DLs provide operators for defining complex concept and role definitions that provide powerful expressive capabilities. Additionally, because of their foundation in First-Order Logic, DLs provide a basis for making logical inferences on the knowledge bases that they define.

Semantic Web technologies apply the mathematical rigor of DLs to the web. The Semantic Web provides a framework that brings structure to web content, provides for unified access to data, and ultimately improves the efficiency of human/computer interactions. It does this through a set of standards (or proposed standards) for defining and using ontologies. The most significant of these are the Resource Description Framework (RDF) and the Web Ontology Language (OWL). Taken together, RDF and OWL fully capture the semantics of the expressive DL, SROIQ, and can be used to describe web content in a semantically rich way.

Among the most significant advantages of the Semantic Web is the ability to computationally reason about ontologies. The most common ontological reasoning tasks

are satisfiability, subsumption, equivalence, disjointness, classification, consistency, instance checking, retrieval, and conjunctive query answering. All of these can ultimately be reduced to a single task—satisfaction. Tableau algorithms are among the most common algorithms for reasoning and are utilized by most well-developed production and research systems. Automata-based algorithms, and resolution-based algorithms are also utilized, and other methods based on First-Order Logic have also been used in applications that are isomorphic to decidable First-Order Logic problems.

Standard reasoning with ontologies (i.e., those tasks that can be reduced to satisfaction) are useful for identifying knowledge that is implicitly contained in the knowledge base. They are not, applicable to the tasks of induction and abduction, however, because these reasoning tasks infer things that might be true from the ground truth of the knowledge base. Machine learning is appropriate for this type of reasoning, and it is not surprising that a number of techniques from this field have been applied to the Semantic Web. Among the most important of these are Inductive Logic Programming and relational matrices (with relational tensors providing a promising area for future research). Instance-Based Learning, kernel functions, and relational graphs have also shown promise in Semantic Web applications.

B. RECOMMENDATIONS FOR FUTURE WORK

Effective utilization of Semantic Web technologies to support situational awareness and decision-making will be facilitated by efforts in a few specific areas. Vocabulary definition is among the most important. The practical difficulty of ontology definition has been mentioned, but the process can be aided by agreement concerning terminology and definitions. Implementation of Semantic Web technologies in the civilian sector has been hampered by a lack of coordination, parallel development, competing interests, and many more factors. Semantic Web implementation for government use, on the other hand, is still in its infancy. A number of efforts are already underway to define domain-specific vocabularies that can be leveraged for Semantic Web applications, and aligning these efforts will facilitate the development of a common vocabulary.

Ontology-development is the next logical step following vocabulary definition. As discussed previously, it is unlikely that a single ontology can be developed that will meet all requirements. Rather, a more reasonable approach is to define ontologies that capture the semantics of specific data sources. This will enable the development of reasonable ontologies that can be incorporated into distributed applications later. The availability of a well-defined common vocabulary will facilitate future incorporation of disparate data sources into applications using the hybrid approach described in Chapter III.

Once Semantic Web content is made available through the development of compatible ontologies for various data sources, application development will be a straightforward proposition. Each application will be able to access and utilize appropriate data sources and maintain its own interpretation of the distributed knowledge base.

Finally, formal product evaluation should be conducted to identify those most capable of supporting DOD knowledge management requirements. A number of existing and forthcoming products are available that will potentially meet envisioned ontology development and maintenance requirements. These products include well-vetted, state-of-the-art algorithms that leverage decades of prior research in DL reasoning. These products and the algorithms they implement are regularly updated as technology evolves.

On the other hand, technologies that leverage Semantic Web content in abductive and inductive reasoning are not currently available in commercial systems. These technologies are likely to be important components of future Semantic Web support for decision-aid systems, however. Although many machine learning techniques can be applied to Semantic Web content, primary consideration must be given to scalability. Of the techniques that have been the subject of significant research, relational matrices and tensors are the most likely to yield results in the near term. Additionally, Instance-Based Reasoning should be explored for the applications to which it is well-suited. Other approaches from (Rettinger, et al., 12) might prove useful at some point, but are less likely to provide significant benefit in the near term.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Allen, J., "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, Vol. 26, No. 11, November 1983.

Apache Software Foundation, Apache Jena online documentation. Available at http://jena.apache.org/index.html (accessed April 2013)

Baader, F. and Sattler, U., "An Overview of Tableau Algorithms for Description Logics", *Proceedings of the TABLEAUX 2000*, St. Andrews, Scotland, UK, July 2000, extended version published by Kluwer Academic Publishers, Dordrecht, Netherlands, 2001.

Baader F., Calvanese, D., McGuinness, D., Nardi, D. (Eds.) and Patel-Schneider, P. (Eds.), Description Logic Handbook: Theory, Implementation and Applications, 2nd Edition, Cambridge Press, 2007.

Bennett, B., "Modal Logics for Qualitative Spatial Reasoning", *Journal of the Interest Group in Pure and Applied Logic*, Vol. 4, No. 1, February 1996.

Berners-Lee, T., Hendler, J., and Lassila, O., "The Semantic Web", *Scientific American*, May 2001.

Boag, S., Chamberlin, D., Fernandez, M., Daniela F., Robie, J. and Simeon, J. (Eds.), "XQuery 1.0: An XML Query Language (2nd Edition), W3C Recommendation, December 2010. Available at http://www.w3.org/TR/xquery/ (accessed February 2013)

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. and Yergeau, F. (Eds.), "Extensible Markup Language (XML) 1.0 (5th Edition), W3C Recommendation, November 2008. Available at http://www.w3.org/TR/2008/REC-xml-20081126/ (accessed February 2013)

Brickley, D. and Guha, R. (Eds.), "RDF Vocabulary Description Language 1.0: RDF Schema", W3C Recommendation, February 2004. Available at http://www.w3.org/TR/rdf-schema/ (accessed February 2013)

Calvanese, D., Eiter, T., and Ortiz, M., "Regular Path Queries in Expressive Description Logics with Nominals", *Proceedings of the 21st International Conference on Artificial Intelligence*, Pasadena, CA, July, 2009.

Calvanese, D., Carbona, D., and Ortiz, M., "A Practical Automata-Based Technique for Reasoning in Expressive Description Logics", *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, Barcelona, SP, July, 2011.

Clark, J. and DeRose, S. (Eds.), "XML Path Language (XPath) Version 1.0", W3C Recommendation, November 1999. Available at http://www.w3.org/TR/xpath/ (accessed February 2013)

Daconta, M. C., Obrst, L. J. and Smith, K. T., *The Semantic Web, A Guide to the Future of XML, Web Services, and Knowledge Management*, Wiley Publishing, 2003.

d'Amato, C., Fanizzi, N., and Esposito., F., "Query Answering and Ontology Population: An Inductive Approach", *Proceedings of the 5th European Semantic Web Conference, Volume 5021 of Lecture Notes on Computer Science*, Springer-Verlag, Berlin Heidelberg, 2008.

Dolby, J., Fokoue, A., Kalyanpur, A., Schonberg, E., and Srivnas, K., "Scalable Highly Expressive Reasoner (SHER)", *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 7, Is. 4, May 2009.

Fanizzi, N. and d'Amato, C., "Inductive Concept Retrieval and Query Answering with Semantic Knowledge Bases through Kernel Methods", *Proceedings of the 11th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Volume 4692 of Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin Heidelberg, 2007.

Franconi, E., ICOM Online Documentation, Free University of Bozen-Bolzano, IT August 2010. Available at http://www.inf.unibz.it/~franconi/icom/ (accessed February 2013)

Gao, S., Sperberg-McQueen, C. and Thompson, H. (Eds.), "W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures", W3C Recommendation, May 2001. Available at http://www.w3.org/TR/xmlschema11-1/ (accessed February 2013)

Getoor, L., Friedman, N., Koller, D., Pferrer, A., and Taskar, B., "Probabilistic Relational Models", in Getoor, L., and Taskar, B. (Eds.), *Introduction to Statistical Relational Learning, MIT Press*, Cambridge, MA, 2007.

Glimm, B., Horrocks, I., and Sattler, U., Deciding SROIQ$^{\cap}$ Knowledge Base Consistency using Alternating Automata", *Proceedings of the 21st International Workshop on Description Logics*, Dresden, Germany, 2008.

Gruber, T. R., "A Translation Approach to Portable Ontologies", Vol. 5, No. 2, April 1993.

Haarslev, V., Hidde, K., Möller, R., and Wessel, M., "RacerPro User's Guide Version 2.0 (draft)", Racer Systems GmbH & Co. KG, October, 2012.

Haarslev, V., Pai, H., Nematollaah, S., "Completion Rules for Uncertainty Reasoning with the Description Logic ALC", Proceedings of the Semantic Web Working Symposium, Quebec, Canada, June 2006.

Hayes, P. (Ed.), "RDF Semantics", W3C Recommendation, February 2004. Available at http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ (accessed February 2013)

Highfleet Semantic Technologies, Integrated Ontology Development Environment online information, 2013. Available at http://www.highfleet.com/iode.html (accessed February 2013)

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean, M. (Eds.), "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", W3C Member Submission, May, 2004. Available at http://www.w3.org/Submission/SWRL/ (accessed February 2013)

Horrocks, I., Kutz, O, and Sattler, U., "The Even More Irresistible $\mathcal{SROIQ}$", *Proceedings of the 10th International Conference on Principles of Knowledge Representation*, Lake District, UK, June 2006.

Horrocks, I., "Ontologies and the Semantic Web", *Communications of the ACM*, Vol. 51, No. 12, December 2008.

Huang, Y., Tresp, V., Nickel, M., Rettinger, A., and Kriegel, H., "A Scalable Approach for Statistical Learning in Semantic Graphs", *Semantic Web*, IOS Press, Pre-press online version, January, 2013. Available at http://iospress.metapress.com/content/u4l35k421324q3l5/?p=0e6f79c18652441bb6bfa36f2535ea3c&pi=6 (accessed March 2013)

Kapoor, B. and Sharma, S., "A Comparative Study of Ontology Building Tools for Semantic Web Applications", *International Journal of Web and Semantic Technology*, Vol. 1, No. 3, July, 2010.

Kashyap, V. "Information Modeling on the Web. The Role of Metadata, Semantics and Ontologies", *Practical Handbook of Internet Computing*, CRC Press, 2004.

Kashyap, V., Bussler, C., Moran, M., *The Semantic Web: Semantics for Data and Services on the Web*, Springer-Verlag, Berlin Heidelberg, 2008.

Kazakov, Y. and Motik, B., "A Resolution-Based Decision Procedure for SHOIQ", *Journal of Automated Reasoning*, Vol. 40, Is. 2-3, March, 2008.

Klyne, G. and Carroll, J. (Eds.), "SPARQL Query Language for RDF", W3C Recommendation, January 2008. Available at http://www.w3.org/TR/rdf-sparql-query/ (accessed February 2013)

Krötszch, M., "OWL 2 Profiles: An Introduction to Lightweight Ontology Languages", *Reasoning Web 2012, Volume 7487 of Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg, 2012.

Krötzsch, M., Simančik, F. and Horrocks, I., "A Description Logic Primer", arXiv:1201.4089v1 [cs.AI], 19 Jan 2012.

Küsters, R and Molitor, R., "Structural Subsumption and Least Common Subsumers in a Description Logic with Existential and Number Restrictions", *Studia Logica: An International Journal for Symbolic Logic*, Vol. 81, No. 2, Nov., 2005.

Lin F. and Sandkuhl K., "A Survey of Exploiting WordNet in Ontology Matching" in IFIP International Federation for Information Processing, Vol. 276, Artificial Intelligence and Practice II, Max Bramer (Ed.), Springer, Boston, MA, 2008.

Maedche, A., "A Machine Learning Perspective for the Semantic Web," *Semantic Web Working Symposium (SWWS) Position Paper*, Palo Alto, CA, July 2001.

Maedche, A. Neumann, G. and Staab, S., "Bootstrapping an Ontology Based Information Extraction System", Szczepaniak, P., Segovia, J. Kacprzyk, J. and Zadeh, L. (Eds.), *Intelligent Exploration of the Web (Studies in Fuzziness and Soft Computing)*, Physica-Verlag, Heidelberg New York, 2003.

Marker, D., *Model Theory: An Introduction*, Graduate Texts in Mathematics, Springer-Verlag, Berlin Heidelberg, 2007.

Motik, B. and Sattler, U., "A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes", *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, Volume 4246 of Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg, 2006.

Motik, B. (Ed.), KAON2 Online Documentation, June 2008. Available at http://kaon2.semanticweb.org/ (accessed March 2013).

Motuk, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A, and Lutz, C., "OWL 2 Web Ontology Language Profiles (Second Edition)", W3C Recommendation, December 2012. Available at http://www.w3.org/TR/owl2-profiles/ (accessed February 2013)

Muggleton, S. and Raedt, L., "Inductive Logic Programming: Theory and Methods", *Journal of Logic Programming*, Vols. 19-10, Sup. 1, May-July 1994.

Ortiz, M., *Query Answering in Expressive Description Logics, Techniques and Complexity Results*, Ph.D. Dissertation, Vienna University of Technology, Vienna, Austria, April 2010.

Ortiz, M. and Šimkus, M., "Reasoning and Query Answering in Description Logics", *Reasoning Web 2012, Volume 7487 of Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg, 2012.

Patel-Schneider, P. and Simeon, J., "The Yin/Yang Web: XML Syntax and RDF Semantics", *Proceedings of the 11th International World Wide Web Conference*, Honolulu, HI, May 2002.

Pease, A. (Ed.), IEEE Standard Upper Ontology Working Group online SUMO documentation, 2004. Available at http://suo.ieee.org/SUO/SUMO (accessed February 2013)

Peterson, D., Gao, S., Malhotra, A., Sperberg, McQueen, C. and THompson, H. (Eds.), "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes", W3C Recommendation, April 2012.

Randell, D., Cui, Z and Cohn A., "A Spatial Logic Based on Regions and Connection", *Proceedings of the 3rd International Conference on the Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, Los Altos, 1992.

Rettinger, A., Lösch, U., Tresp, V., d'Amato, C., and Fanizzi, N., "Mining the Semantic Web", *Data Mining and Knowledge Discovery*, Vol. 24, Is. 3, May 2012.

Rudolph, S., "Foundations of Description Logics, Reasoning Web: Semantic Technology for the Web of Data", *7th International Summer School, Volume 6848 of Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg, 2011.

Russell, S. and Norvig, P., *Artificial Intelligence, a Modern Approach*, Third Edition, Prentice Hall, 2010.

Sandpiper Software, Visual Ontology Modeler online information, 2013. Available at http://www.sandsoft.com/products.html (accessed February 2013)

Schmidt-Schaub, M. and Smolka, G., "Attributive Descriptions with Complements", *Artificial Intelligence*, Vol. 48, No. 1, February 1991.

Semafora Systems, *OntoStudio Version 2.1*, 2008. Available at http://www.semafora-systems.com/fileadmin/user_upload/Produktdoku_EN/Productdocumentation_OntoStudio_2.1_en.pdf (accessed February 2013)

Shvaiko, P. and Euzenat, J., "Ten Challenges for Ontology Matching", Proceedings of the 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE), Monterrey, MX, November 2008.

Shvaiko, P. and Euzenat, J., Ontology Matching: State of the Art and Future Challenges", IEEE Transactions on Knowledge and Data Engineering, Vol. 25, No. 1, January 2013.

Singh, A., *Efficient Matrix Models for Relational Learning*, Ph.D. Dissertation, Carnegie Mellon University, October 2009.

Singh, S and Karwayun, R., "A Comparative Study of Inference Engines", *Proceedings of the Seventh International Conference on Information Technology*, Las Vegas, NV, April 2010.

Sirin, E., Parsia, B., and Grau, B., Aditya Kalyanpur and Yarden Katz, "Pellet: A Practical OWL-DL Reasoner", *Journal of Web Semantics*, Vol. 5, Is. 2, June 2007.

Stanford University Center for Biomedical Informatics Research (CBIR), Protégé Online Documentation. Available at http://protege.stanford.edu (accessed February 2013)

Stanford University Knowledge Systems, AI Laboratory (KSL), Ontolingua online documentation, 2013. Available at http://www.ksl.stanford.edu/software/ontolingua/ (accessed February 2013)

Straccia, U., "Reasoning with Fuzzy Description Logics", Journal of Artificial Intelligence Research, Vol. 14, April 2001.

Trest, V., Bundschus, M., Rettinger, A., and Huang, Y., "Towards Machine Learning on the Semantic Web", *Uncertainty Reasoning for the Semantic Web I, Volume 5327 of Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg, 2008.

Tsarkov, D. and Horrocks, I., "FaCT++ Description Logic Reasoning System Description", *Third International Joint Conference on Automated Reasoning, Volume 4130 of Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg, 2006.

Vidya, V. and Punitha, S., "A Survey on Ontology Tools", *International Journal on Scientific and Engineering Research*, Vol. 3, No 10, October 2012.

W3C OWL Working Group (Eds.), "OWL 2 Web Ontology Language Document Overview (2nd Edition)", W3C recommendation, December 2012. Available at http://www.w3.org/TR/owl2-overview/ (accessed February 2013)

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California